BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2019 Bc. Daniel Adámek

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

AUTOMATIC CALIBRATION OT ROBOTIC ARM USING CAMERAS
AUTOMATICKÁ KALIBRACE ROBOTICKÉHO RAMENE POMOCÍ KAMER/Y

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Daniel Adámek**
AUTOR PRÁCE

SUPERVISOR — **Ing. Peter Honec, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Daniel Adámek ***ID:*** 159199
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Automatická kalibrace robotického ramene pomocí kamer/y

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s robotickým systémem a nynějším řešením manuální kalibrace.
2. Seznamte se s problematikou a metodami 3D rekonstrukce.
3. Proveďte analýzu možných řešení a navrhněte vhodnou metodu.
4. Vyberte vhodné optické vybavení.
5. Implementujte vybrané řešení v prostředí .NET.
6. Otestujte a vyhodnoťte funkčnost.

DOPORUČENÁ LITERATURA:

Kanatani Kenichi, Sugaya Yasuyuki, Kanazawa Yasushi: Guide to 3D Vision Computation, ISBN 978-3-3-9-48492-1

HLAVAC V., SONKA M., BOYLE R.: Image Processing, Analysis, and Machine Vision, ISBN 978-0495082521

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 13.5.2019

Vedoucí práce: Ing. Peter Honec, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRACT

To eliminate human factor in end-to-end testing of embedded devices, it is needed to develop complex robotic automated system. One of crucial tasks is to automatically calibrate such a system. In this thesis I analyzed multiple approaches how to estimate robot-device spatial relation using camera/s and presented solution based on one camera pose estimation using iterative methods such as Gauss-Newton or Levenberg-Marquardt. At the end, I discussed the accuracy of the solution and introduced possible improvements and the direction of next development.

KEYWORDS

Pose estimation, iterative method, marker detection, Gauss-Newton, robot arm calibration

ABSTRAKT

K nahrazení člověka při úloze testování dotykových embedded zařízení je zapotřebí vyvinout komplexní automatizovaný robotický systém. Jedním ze zásadních úkolů je tento systém automaticky zkalibrovat. V této práci jsem se zabýval možnými způsoby automatické kalibrace robotického ramene v prostoru ve vztahu k dotykovému zařízení pomocí jedné či více kamer. Následně jsem představil řešení založené na estimaci polohy jedné kamery pomocí iterativních metod jako např. Gauss-Newton nebo Levenberg-Marquardt. Na konci jsem zhodnotil dosaženou přesnost a navrhnul postup pro její zvýšení.

KLÍČOVÁ SLOVA

Určení pozice v prostoru, iterativní metoda, detekce markeru, Gauss-Newton, kalibrace robotického ramene

ADÁMEK, Daniel. *Calibration of Robot Arm using Camera*. Brno, 2019, 72 p. Master's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Advised by Ing. Peter Honec, PhD.

DECLARATION

I declare that I have written the Master's Thesis titled "Calibration of Robot Arm using Camera" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Peteru Honcovi, Ph.D. za odborné vedení, konzultace, podnětné návrhy k práci, rum a historky z mládí. Dále robotickému týmu za možnost tuto práci pod firmou YSoft vypracovat, jmenovitě Kubovi Pavlákovi za nekonečnou trpělivost s mou (ne)dochvilností a nočními pracovními návyky. A v neposlední řadě panu Ing. Janu Klečkovi za nespočet odborných konzultací.

Brno ....

author's signature

Contents

List of Figures

List of Tables

Introduction

Most of today's software companies must be thinking about testing and balance safety (and stability) of their application and costs of achieving it in the best possible manner for their product. Some products are on such a high level of complexity that it is necessary to implement end to end testing, meaning to test the application also on given hardware. These tests have been performed mostly by human testers for their automation was economically inconvenient, at least to the recent times. To automate such a high-level operation, you need to put together pieces from variant fields such as computer vision, AI, robotics, sensors etc.

My task in this thesis is to automatically calibrate such a system which is composed of a robot arm and a camera and is testing application running on devices with touch screen. This means to find solution (software and hardware) which is able to find touch screen in the scene and then to tell where the screen is in the coordinate system of the robot arm so the camera system could search for icons, texts and so and the robot arm could perform clicks, swipes etc. This I should achieve with given system. In case it is not possible I am expected to propose changes on this system which allows automatic calibration. These changes must also be reasonable from economical point of view and must respect the architecture of the present system (program languages, database). This piece of puzzle helps human operators to set the whole system without manual calibration which also helps to black-box the whole high-end testing even more. Additionally, it allows later development of mobile robots which would be able to find given device by itself so the human operator could be skipped in the whole process.

This thesis is done for the YSoft company so the whole solution was designed and tested for a real industrial problem.

1 Analysis of assignment and set up

In this chapter I explain how the calibration was done before and specify conditions under which is the system supposed to work with automatic calibration. After that the analysis of several approaches is provided and at the end of the chapter I present conclusion which approach I chose to describe in detail and implement.

1.1 Description of the system

The system consists of custom robot arm with reach distance approximately 40cm. The camera is Basler acA 1920-40gc with HF25HA-1B lens.

Specification of camera and lens

- resolution: 1920×1080 pixels
- pixel size: 5.86×5.86µm
- focal length: 25mm
- aperture: 1.4

The camera and the robot arm are attached to tripod, each to its own, meaning they are not mechanically fixed. The main focus of the YSoft testing is their interface installed on big office printers with touch screens. Besides that the company has its own terminals (also with touch screens) and its smartphone application which both are also supposed to be tested.

Before the automatic one, the calibration was done manually. It was divided into 2 procedures.

1.1.1 Calibration of camera view

The camera is set perpendicularly above the screen and the corners are detected using Canny detector. Then user has to mark the corners of the screen in the user interface, and the nearest detected corner is assigned as screen corner. These corners are used for cropping the raw image from a camera.

1.1.2 Calibration of robot arm

Robot arm is calibrated also by user by displaying calibration view on the device screen. Then user navigates robot arm (by keyboard) to certain points on the screen. From the data from printer screen and the data about position of robot motors the system can determine transformation from robot arm coordinate system to screen coordinate system.

So, it is not just about need of a human tester to calibrate the system, it is also about getting data from the tested device.

1.2 Assignment and conditions

The main reason why to implement automatic calibration is to get rid of the human element, since this part of using the robotic testing system was evaluated as crucial in terms of utilization by developers. Also, there is the tendency to black-box it even more so the system would not need connection with tested device.

Present system can provide these inputs

- images from camera
- tested device ID
- screen size of the device
- camera and robot arm IDs

The rest of the properties of environment and the device which will prove to be necessary must be collected and the database must be created.

There are several general conditions which are desirable to meet (more or less).

- Do not alter tested device (printer, etc.) if possible
- Must work in different lighting conditions
- Must work with as much device types as possible
- Cannot depend on a state of a device (on/off/idle...)

As you will see, it is almost impossible to fulfill all those presumptions in reasonable time and costs, therefore compromises are needed.

1.2.1 Tested devices

There are 3 main types of devices - printers, YSoft terminals and smartphones. In account of the task, these properties must be taken into consideration

- type of a touch screen (resistive/capacitive)
- size of the screen
- recessing of the screen
- distance of hardware buttons and edges
- flatness of the screen and the surrounding

Since the main product of YSoft is linked with printers, the solution will consider mainly use of printers as tested devices.

2 Analysis of approach

In this chapter I will introduce several approaches how to detect screen of tested device and how to determine a position of the device screen in the robot arm coordinate system. Some of the approaches solve both problems, some of them solve only one of them so it must be combined with complementary approach. I will not provide detailed theoretical description but only compare the advantages and disadvantages in account of the task and discuss the difficulty of implementation and performance at the end of the chapter. Also I will choose the approach which I will describe theoretically and implement as final solution.

2.1 2D active triangulation

This approach needs one camera and a projector in one optical system with known relative positions between them and is able to solve both tasks - screen detection and relative position of screen and robot arm. The projector (for example linear laser) projects given pattern on measured object. Then it is possible to get the distance of the object and camera from the known proportions of the object and the deformation of the pattern ([1]). The main trouble with reliability here is to detect the pattern precisely.

To get requested accuracy I would have to use a field of focused laser crosses. There are several challenges which make detection problematic

- Can use maximum 5mW laser
- Overlapping of the laser wave length and the emission spectrum of the screen
- Resistive touch screens have 2 layers - meaning 2 possible reflections
- Screen can get dust on its surface - different reflection

Since the problem with double reflection in case of the resistive screen can be solved by known angle of projection - at least knowledge if it is projected from the left/right side from the normal of screen plane - and also the case of the dust or other impurity of the screen could be solved by interpolation (if the pollution would have local character), the main problem seems to be the fact, that the automatic calibration will be used during execution of tests - meaning the screen will be on. There is always possibility to switch the printer off but it would increase time demands of whole procedure. To find out if it is possible to detect laser reflection from the screen which is switched on, emission spectrum measurement must be done.

For the analysis I used 5mW red laser cross with 650nm wave length and measured spectrum of a screen (on/off) with and without the exposure of laser cross.

The figures above are chosen to show the case when the screen is also emitting red color (approximately the same wave length) so the detection seems to be very

Fig. 2.1: Red linear 650nm laser on a switched-on screen

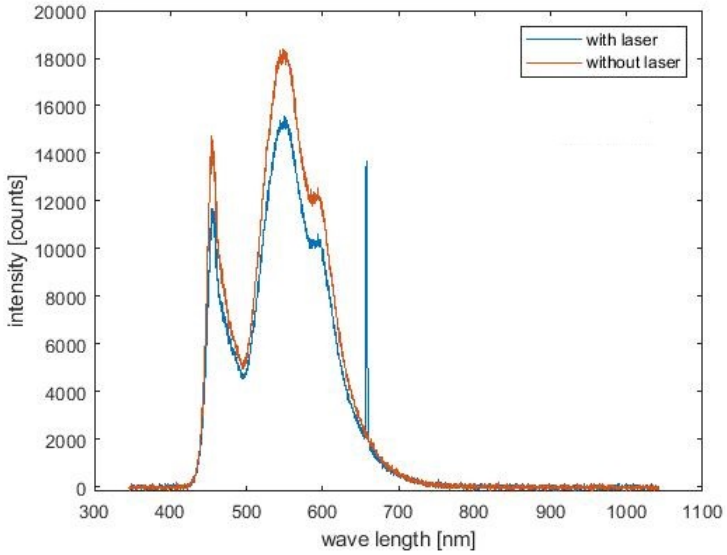

Fig. 2.2: Emission spectrum of OKI MC873 printer without and with laser

problematic because the red LED detects all wave lengths above 570 approximately. Another solution is to use a wave length, which is not that dominant in the emitting spectrum of the LCD ([13]). This could be, for example, the 405nm wave length, which is still widely available. Still there is need to use narrow-band optical filter to obtain the reflection (it is detected by the blue LED which detects wave lengths approximately from 550nm down to 300nm - [13]).

But maybe the biggest disadvantage is that this approach does not work if the

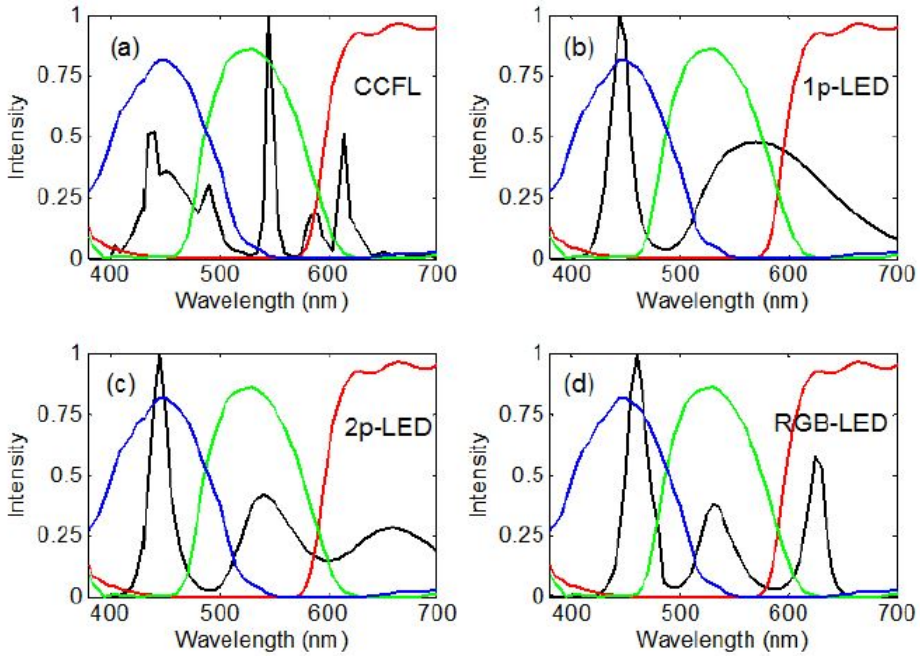

Fig. 2.3: Normalized emission LCD spectra of four light sources (black curves) and color filters (RGB curves), ([13])

screen is not recessed into the front panel of printer.

2.2 Linear triangulation

Reconstructing the space in the front of the optical system using linear triangulation requires 2 cameras in known relative position. The main idea is to find the same structures at pictures from both cameras and then from projection matrices of cameras and found correspondences at pictures it is possible to get the relative position of measured device and the cameras ([9]).

Big advantage is that it is not necessary to know the mathematical model of found structure on the device. So, it is possible to use for example screen detection algorithm based on edge detection without actual knowledge about screen size. One disadvantage is obvious - need of the second camera therefore higher price. The second disadvantage is mechanically fixed cameras.

2.3 One camera estimation

To determine relative position of the measured object and the camera it is necessary to detect points on the object and also to know their mutual spatial relationship - mathematical model. If those two things are obtained, then using numerical iteration methods for nonlinear problems such as Gauss-Newton method or Levenberg-Marquardt method it is possible to reconstruct the points to 3D, therefore get the position of screen.

2.4 Marker detection

Maybe the most robust and, at the first glance, the easiest to implement way how to detect the printer screen could seem the marker detection. It can be done by template matching, searching for features in image and compare those with features of the marker and other different approaches.

Big advantages are perfect knowledge of the searched object, possibility to design the marker and to decide its position. Disadvantages are necessity to alter tested device as well as possibility of damage of the marker and its maintenance.

2.5 Edge detection

Since the searched object is rectangular, it is straight forward to think about its detection in terms of edge detecting algorithm. By methods as Canny detector or using Hough transformation, it is possible to obtain edge representation of the image and then find the screen by set of rules derived from a priori knowledge (like ratio between screen sizes).

The biggest advantage is that it is not necessary to alter measured object. But there are two main disadvantages - firstly it is the fact that the camera is set by a human tester and in the case of recessed screen there is a possibility that part of the screen will not be seen by camera therefore it will detect incomplete screen which leads to wrong estimation of screen position in 3D. The second problem is caused by variety of all possible images which can occur on the screen during tests. Some of those screens contain rectangular pop-up frames almost as big as the whole screen and they are displayed on a dark background causing false detections of a pop-up frame as the whole touch screen.

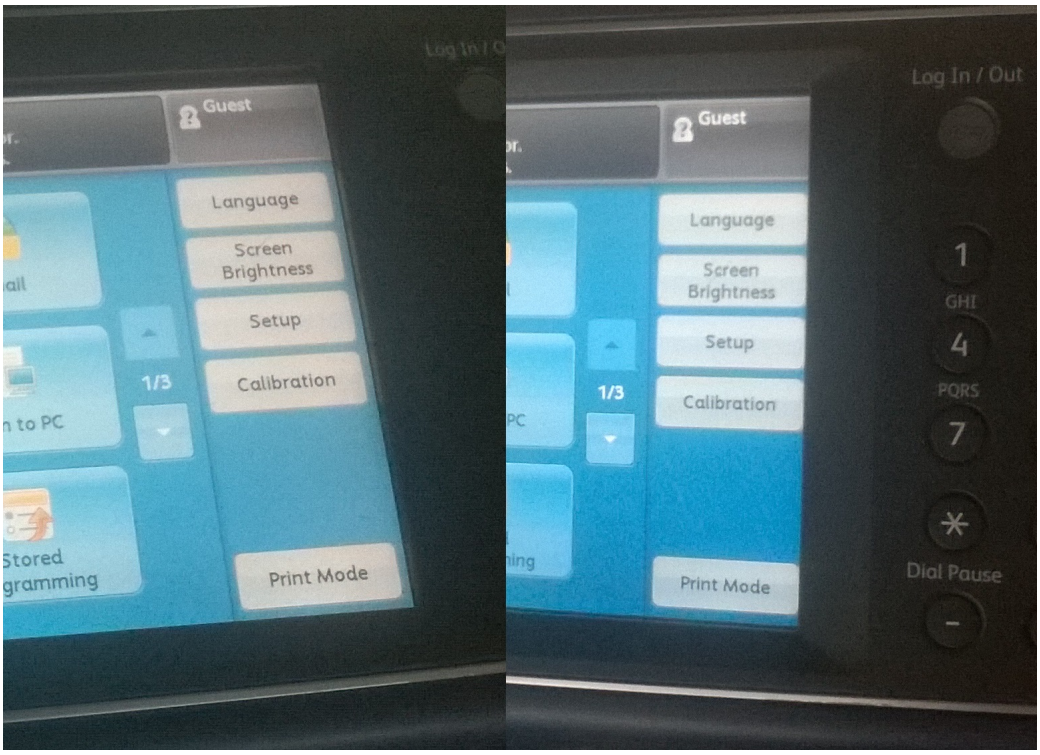

Fig. 2.4: Approximately 2mm of screen might not be seen by camera due to recessing of the touch screen and setting the camera

2.6 Robot and camera setting

All those approaches how to detect touch screen of a device or how to determine the relative position of the screen and the testing system have one thing in common. They are all able to find the relationship only between the camera and measured object. So, there are two possible solutions how to get the information between transformation from the robot arm to the object.

2.6.1 Mechanically fixed camera and robot

First is to connect the whole system in one piece of hardware so the transformation from the camera to the robot arm is known. The problem is in a number of different devices with a different screen size. To maximize the system accuracy, it is desirable to fill the whole image with the screen. This can be done by moving the camera up and down (setting the tripod) or by changing lenses. Since every time the lens is changed it is necessary to re-calibrate the camera, it is much more convenient to move the camera. Of course, there is a possibility to have set of cameras with different lenses attached but it is again much more expensive solution then to have

just one universal combination camera-lens. Or it would be possible to have set of hardware frames connecting the robot arm and the camera but again - it is much more expensive and impractical.

2.6.2 No physical connection

The second option is to use detection not only for the screen, but also for the robot arm. Therefore, there is no need to connect it firmly, but the robot arm must be marked. Since the arm is a part of the system, it can be altered in much bigger scale. The whole arm can be covered by chessboard pattern or a marker can be placed on the base plane of the robot. But the relationship between marker (or chessboard) must be known (which should not be the problem). So, there is a big advantage of the variability of the system but also big disadvantages - the accuracy is lower and the robot arm (or at least its marker) must be seen by the camera.

2.7 Conclusion

After a discussion with the YSoft team the order of priorities was set in this way

1. Number of devices for which it can be used
2. One version of the system (not a lot of sets of cameras + lenses or hardware frames)
3. Price

Considering this order, I decided to work on the version with finding the screen by marker detection and using only one camera and known model of the marker to determine the position of the printer screen. Since the request of not altering the measured device turned out to be less important, the marker detection is the most robust approach. Secondly, the decision about using one camera and iterative process of getting the position is based on fact that I will have the information about the size of the screen. Then I can generate database of mathematical models using just one type of marker and change the size of a marker with the screen size. In terms of hardware connection of a camera and a robot arm, I chose to work on the version without the hardware frame due to second point in the order.

3 Theory

In this chapter I will describe theory of chosen approaches.

3.1 Basics

Firstly, it is necessary to introduce some basics to understand further theory.

3.1.1 3D projection

The 3D projection is a mapping of 3D point to 2D plane (e.g. of a sensor) done by projection matrix $\mathbf{P}$ with size 4×3.

$$\mathbf{x} = \mathbf{PX} \tag{3.1}$$

where $\mathbf{x}$ is a projected 2D point vector, $\mathbf{P}$ is a projection matrix and $\mathbf{X}$ is vector of original 3D points. The projection matrix can be obtained from intrinsic parameters of a camera and from its position

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{3.2}$$

where

- $\mathbf{t}$ is vector of translation
- $\mathbf{R}$ is a rotation matrix defined as $\mathbf{R} = \mathbf{R_z}\mathbf{R_y}\mathbf{R_x}$, where $\mathbf{R_z}$, $\mathbf{R_y}$, $\mathbf{R_x}$ are rotations around axes z, y, x respectively.
- $\mathbf{K}$ is intrinsic matrix of a camera

3.1.2 3D transformation

Transformation is similar to projection, only without the effect of losing one dimension. In this task it is possible to work with just basic 3D transformations like rotation and translation. Then transformation matrix $\mathbf{T}$ (4×4) is defined as

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.3}$$

3.1.3 Homogeneous coordinates

Both projection and transformation use homogeneous coordinates. It is cartesian coordinate with 1 extra scaling dimension. So, e.g. cartesian coordinate $[1, 0, 1]$ is represented in homogeneous coordinate as $[1, 0, 1, 1]$. Also, since the last component is a scale, it leads to an equality

$$[a, b, c, s] = [a/s, b/s, c/s, 1] \tag{3.4}$$

Transformation from homogeneous to cartesian is done by dividing the homogeneous coordinate by scaling factor and cutting of the last dimension (scale).

3.1.4 Jacobian matrix

Jacobian matrix is the matrix of first-order partial derivatives of a vector function. If we have a function f which takes the vector $\mathbf{p}$ as input and returns the vector $f(\mathbf{p})$ as output, then Jacobian matrix is defined as

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial p_1} & \cdots & \frac{\partial f_1}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial p_1} & \cdots & \frac{\partial f_m}{\partial p_n} \end{bmatrix} \tag{3.5}$$

for $f(\mathbf{p})$ of size m and $\mathbf{p}$ of size n.

3.1.5 Logistic function

To improve marker detection, I use logistic function (also called sigmoid) with equation

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} + d \tag{3.6}$$

, where

- L is curve maximum value
- x_0 is sigmoid's midpoint x-value
- k is steepness of the curve.
- d is shift along x-axis

With this curve I approximate rising edges of corner shape.

3.1.6 Least Square Fitting Line

Method FitLine from OpenCV library uses linear regression - it approximates set of points by line using least squares method. Let assume we have line represented as

$$y = k_1 u + k_0 \tag{3.7}$$

then I get k_1 and k_0 coefficients as

$$\begin{aligned} k_1 &= \frac{n \sum_i u_i y_i - \sum_i u_i \sum_i y_i}{n \sum_i u_i^2 - (\sum_i u_i)^2} \\ k_0 &= \frac{\sum_i u_i^2 \sum_i y_i - \sum_i u_i \sum_i u_i y_i}{n \sum_i u_i^2 - (\sum_i u_i)^2} \end{aligned} \tag{3.8}$$

which minimizes squares of distance between points and approximated line (by [14]).

3.1.7 Camera higher distortion models

OpenCV provides three camera distortion models - basic consisting of tangential and 6-th order radial model, rational and thin prism model. I describe first two of them in this section and provide references for the third one.

Tangential and radial basic model

OpenCV uses five higher order distortion coefficients for basic camera model. Three as radial factors

$$\begin{aligned} x_{distorted} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{distorted} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3.9}$$

and two as tangential factors

$$\begin{aligned} x_{distorted} &= x + (2p_1 xy + p_2(r^2 + 2x^2)) \\ y_{distorted} &= y + (p_1(r^2 + 2y^2) + 2p_2 xy) \end{aligned} \tag{3.10}$$

by [8].

Rational function model

Introduced by Claus and Fitzgibbon, rational function model extends standard camera calibration matrix

$$\mathbf{d}(i,j) = \begin{bmatrix} A_{11}i + A_{12}j + A_{13}1 \\ A_{21}i + A_{22}j + A_{23}1 \\ A_{31}i + A_{32}j + A_{33}1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \tag{3.11}$$

, where $\mathbf{A}$ is 3×3 matrix, by mapping which permits i and j to appear in higher order polynomials:

$$\mathbf{d}(i,j) = \begin{bmatrix} A_{11}i^2 + A_{12}ij + A_{13}j^2 + A_{14}i + A_{15}j + A_{16} \\ A_{21}i^2 + A_{22}ij + A_{23}j^2 + A_{24}i + A_{25}j + A_{26} \\ A_{31}i^2 + A_{32}ij + A_{33}j^2 + A_{34}i + A_{35}j + A_{36} \end{bmatrix} \tag{3.12}$$

([10]), which can be written as a linear combination of 3×6 matrix $\mathbf{A}$ and 6-vector of parameters.

Thin Prism Model

The third used distortion model is a thin prism model. For more details about this type I will redirect the reader to [11] or [12].

3.2 Marker Detection

In this section I describe methods and algorithms used for marker detection further in the text.

3.2.1 Conversion from RGB to grayscale

In RGB representation, every pixel is a 3D vector containing values of red, blue and green color. If we want to convert it to grayscale, which is a scalar representing a gray level, we have to form a function. There are two ways how to approach this

First one is a simple average from those three components of RGB.

$$f_{gray}(i,j) = \frac{f_{rgb}(i,j,red) + f_{rgb}(i,j,green) + f_{rgb}(i,j,blue)}{3} \tag{3.13}$$

where $f_{gray}(i,j)$ is output image function on position i, j and f_{rgb} is input RGB image on position i, j.

The second approach is to weight contributions of RGB components.

$$f_{gray}(i,j) = w_r f_{rgb}(i,j,red) + w_g f_{rgb}(i,j,green) + w_b f_{rgb}(i,j,blue) \tag{3.14}$$

where w_r, w_g, w_b are weights and their sum must be equal to 1.

3.2.2 Binarization

Once I have a grayscale image, I can convert it to binary image by setting the threshold. Then every pixel in image is define as

$$f_b = \begin{cases} 1, & \text{if } f_{gray} > threshold \\ 0, & \text{if } f_{gray} \leq threshold \end{cases}$$

There are various methods how to set the threshold - searching for local minima in histogram, Otsu method or methods based on a priori knowledge (like an average ratio between blank and ink covered space on a list of paper).

3.2.3 Contours

After binarization of an image, I search for contours. It is done by comparing each 1-value pixel with its 4- or 8-neighborhood. If there is a 0-value pixel, it is claimed as border pixel. After that all contours represent borders of binary objects in the image and are organize to a hierarchy which I will not use. ([3])

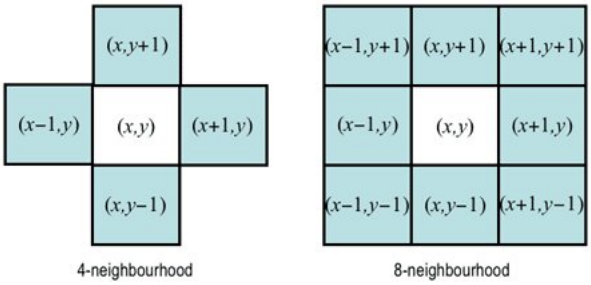

Fig. 3.1: 4- and 8-connected pixels. ([2])

3.2.4 Linearity of a point array

During the detection I have to find a contour with the most linear edges (since it is irregular concave pentagon) between known vertices. I set the metric as root mean square error (RMSE) of distances between points and a line connecting the vertices. Suppose I have a line l defined as $ax + by + c = 0$ and point in 2D $A[a_1, a_2]$, then the distance is

$$d(A, l) = \frac{|aa_1 + ba_2 + c|}{\sqrt{a^2 + b^2}} \tag{3.15}$$

and the RMSE is defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} d(A_i, l)^2}{n}} \tag{3.16}$$

where n is the number of points in an array and $d(A_i, l)$ is distance between i-th point and line l.

3.2.5 Douglas-Peucker algorithm

This algorithm is used in approxPolyDP method from OpenCV library to simplify polygon or curve consisting of set of points. It has one parameter ϵ. The algorithm goes recursively like this

1. Set distance ϵ
2. Take first and last point and connect them with line
3. Find point furthest from the line
4. If the point is closer then ϵ, return first and last point
5. Else call recursively with first and furthest point segment and last and furthest point segment

It can be transformed to version with different parameter - number of polygon vertices. Then it iteratively sets ϵ to get desired polygon.

3.3 Position estimation

To determine the position of the touch screen in the coordinate system of camera I have to use one of iterative estimation methods.

3.3.1 Gauss-Newton iteration

Let's have a function of a form

$$\mathbf{x} = f(\mathbf{p}) \tag{3.17}$$

where $\mathbf{p}$ is a parameter vector, and $\mathbf{x}$ is a measurement vector which approximate true value $\overline{\mathbf{x}}$ (in my case it is vector of detected points of a marker). Then my goal is to find such a vector $\widehat{\mathbf{P}}$ which minimizes equation

$$\mathbf{x} = f(\widehat{\mathbf{P}}) - \epsilon \tag{3.18}$$

over $\|\epsilon\|$. At the beginning we need to estimate initial parameter vector $\mathbf{p_0}$ which is as close to the real $\mathbf{p}$ as possible. Then we get ϵ_0 defined as $\epsilon_0 = f(\mathbf{p_0}) - \mathbf{x}$. By assumption that function f at $\mathbf{p_0}$ is approximated by $f(\mathbf{p_0} + \Delta) = f(\mathbf{p_0}) + \mathbf{J}\Delta$, where $\mathbf{J}$ is Jacobian of function f, so $\mathbf{J} = \partial f / \partial \mathbf{p}$. Then we seek a vector $f(\mathbf{p_1})$, where $\mathbf{p_1} = \mathbf{p_0} + \Delta$, which minimizes ϵ_1. Similar to ϵ_0

$$\epsilon_1 = f(\mathbf{p_1}) - \mathbf{x} \tag{3.19}$$

then by $\mathbf{p_1} = \mathbf{p_0} + \Delta$, approximation of $f(\mathbf{p_0} + \Delta)$ and eq.3.18 we get

$$\begin{aligned} \epsilon_1 &= f(\mathbf{p_0} + \Delta) - \mathbf{x} \\ \epsilon_1 &= f(\mathbf{p_0}) + \mathbf{J}\Delta - \mathbf{x} \\ \epsilon_1 &= \epsilon_0 + \mathbf{J}\Delta \end{aligned} \tag{3.20}$$

which is a linear equation, so we got linear minimization problem. Then we want to minimize its absolute value - $\|\epsilon_0 + \mathbf{J}\Delta\|$ - over Δ.

Similarly to procedure of getting $\mathbf{p_1}$ from a vector $\mathbf{p_0}$, we can generalize it to

$$\mathbf{p_{i+1}} = \mathbf{p}_i + \Delta_i \tag{3.21}$$

where Δ_i is the solution to $\mathbf{J}\Delta_i = -\epsilon_i$

Since the $\mathbf{J}$ is not square matrix (in general), we cannot use a simple inverse to get Δ. Besides using pseudo-inverse, we can get the square matrix by multiplying both sides of equation by $\mathbf{J^T}$

$$\mathbf{J^T}\mathbf{J}\Delta_i = -\mathbf{J^T}\epsilon_i \tag{3.22}$$

And now we can get Δ

$$\Delta_i = -(\mathbf{J^T}\mathbf{J})^{-1}\mathbf{J^T}\epsilon_i \tag{3.23}$$

3.3.2 Levenberg-Marquardt

The Levenberg-Marquardt method is similar as the Newton one. The difference is in calculating Δ

$$\Delta = -(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T\epsilon \tag{3.24}$$

, where $\mathbf{I}$ is identity matrix and λ changes between iteration. The initial value of λ is typically 10^{-3} times the average of the diagonal elements of $\mathbf{J}^T\mathbf{J}$ (by [9]). If the obtained Δ reduces the error ϵ, the coefficient λ is divided by a factor (e.g. 10) for the next iteration. In opposite case when the Δ leads to bigger error, then λ is multiplied by the factor and Δ is calculated again with new λ. The process is repeated within one iteration until the Δ reduces the error. The advantage of this method is that it always changes Δ in a right direction of reducing the error. It is faster if the initial parameter vector $\mathbf{p_0}$ is far from the real $\mathbf{p}$ but slower if the initial guess is close.

3.4 My case

3.4.1 3D Reconstruction

I need to define several variables so I can use Newton iterative method (or similar) to reconstruct detected points using their model, such as function $f(\mathbf{p})$ or Jacobian matrix $\mathbf{J}$.

parameters p

We need to begin with the desired result which is position of a screen in the coordinate system of camera or in the mathematical point of view - transformation matrix from coordinate system of screen to coordinate system of a camera. That is transformation 3D $\rightarrow$ 3D given by matrix 4×4. Let's say that the origin of a global coordinate system is a touch screen (e.g. its left down corner). Then I know 2 things

- position of all marker's points in this system
- position of all projections of these points in camera's image

Then I can ask the question - what is the projection matrix of camera which gives me such projections of known 3D points? Or in other way - what is the position of the camera which gives me that projection matrix? When I know the projection matrix I also know the position of camera in coordinate system of the screen, therefore I can derive transform matrix. Here I get the first variable of the equation - parameter vector $\mathbf{p}$ given as

$$\mathbf{p} = \begin{bmatrix} t_x & t_y & t_z & \alpha & \beta & \gamma \end{bmatrix}^T \tag{3.25}$$

where t_x, t_y, t_z are components of translation vector $\mathbf{t}$ and α, β, γ are angles of rotation around axes x, y, z giving me rotation matrix $\mathbf{R}$. Then I have the parts to derive projection matrix as in eq.3.2 (intrinsic matrix must be known).

function f

I also suggested a little what the function f will look like. I have to construct the projection matrix from these parameters so I can see how close the projection is in iteration i to the real projected points so I can get the error ϵ_i. Let's have the projection matrix $\mathbf{P}_i$ as in eq.3.2, then I get vector f_h (as homogeneous vector)

$$f_h(\mathbf{p}) = \begin{bmatrix} x \\ y \\ s \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R_1}\mathbf{X} + t_x \\ \mathbf{R_2}\mathbf{X} + t_y \\ \mathbf{R_3}\mathbf{X} + t_z \end{bmatrix} \tag{3.26}$$

where $\mathbf{R_i}$ is a i-th row of the rotation matrix $\mathbf{R}$ and $\mathbf{X}$ is vector of original 3D points. If I want to compare it with 2D coordinates of camera projection, I have to transform those homogeneous coordinates back to cartesian. Therefore, the final form of a function f is

$$f(\mathbf{p}) = \mathbf{K} \begin{bmatrix} \frac{x}{s} \\ \frac{y}{s} \end{bmatrix} \tag{3.27}$$

where $s = \mathbf{R_3}\mathbf{X} + t_z$ by eq.3.26. This gives me vector $[x_n, y_n]$ of normalized cartesian coordinates.

jacobian of the function

The last thing I have to derive is the Jacobian $\mathbf{J}$ of the function f.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_n}{\partial t_x} & \frac{\partial x_n}{\partial t_y} & \frac{\partial x_n}{\partial t_z} & \frac{\partial x_n}{\partial \alpha} & \frac{\partial x_n}{\partial \beta} & \frac{\partial x_n}{\partial \gamma} \\ \\ \frac{\partial y_n}{\partial t_x} & \frac{\partial y_n}{\partial t_y} & \frac{\partial y_n}{\partial t_z} & \frac{\partial y_n}{\partial \alpha} & \frac{\partial y_n}{\partial \beta} & \frac{\partial y_n}{\partial \gamma} \end{bmatrix} \tag{3.28}$$

which is equal to

$$\begin{bmatrix} \frac{\frac{\partial x}{\partial t_x}s - x\frac{\partial s}{\partial t_x}}{s^2} & \frac{\frac{\partial x}{\partial t_y}s - x\frac{\partial s}{\partial t_y}}{s^2} & \frac{\frac{\partial x}{\partial t_z}s - x\frac{\partial s}{\partial t_z}}{s^2} & \frac{\frac{\partial x}{\partial \alpha}s - x\frac{\partial s}{\partial \alpha}}{s^2} & \frac{\frac{\partial x}{\partial \beta}s - x\frac{\partial s}{\partial \beta}}{s^2} & \frac{\frac{\partial x}{\partial \gamma}s - x\frac{\partial s}{\partial \gamma}}{s^2} \\ \\ \frac{\frac{\partial y}{\partial t_x}s - y\frac{\partial s}{\partial t_x}}{s^2} & \frac{\frac{\partial y}{\partial t_y}s - y\frac{\partial s}{\partial t_y}}{s^2} & \frac{\frac{\partial y}{\partial t_z}s - y\frac{\partial s}{\partial t_z}}{s^2} & \frac{\frac{\partial y}{\partial \alpha}s - y\frac{\partial s}{\partial \alpha}}{s^2} & \frac{\frac{\partial y}{\partial \beta}s - y\frac{\partial s}{\partial \beta}}{s^2} & \frac{\frac{\partial y}{\partial \gamma}s - y\frac{\partial s}{\partial \gamma}}{s^2} \end{bmatrix} \tag{3.29}$$

As it can be seen, I need partial derivatives of x, y, and s (meaning f_h) by parameter vector $\mathbf{p}$. I state just the results of derivatives of f_h by components of $\mathbf{p}$, because that is how I calculate it in the algorithm (and then I take rows as results for x, y, s).

$$\begin{aligned}
\frac{\partial f_h}{\partial \alpha} &= \mathbf{K}[R_z R_y \frac{dR_x}{d\alpha}]\mathbf{X} \\
\frac{\partial f_h}{\partial \beta} &= \mathbf{K}[R_z \frac{dR_y}{d\beta} R_x]\mathbf{X} \\
\frac{\partial f_h}{\partial \gamma} &= \mathbf{K}[\frac{dR_z}{d\gamma} R_y R_x]\mathbf{X}
\end{aligned} \tag{3.30}$$

$$\begin{aligned}
\frac{\partial f_h}{\partial t_x} &= \mathbf{K} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\
\frac{\partial f_h}{\partial t_y} &= \mathbf{K} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\
\frac{\partial f_h}{\partial t_z} &= \mathbf{K} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\end{aligned} \tag{3.31}$$

3.4.2 Sigmoid fitting

To approximate rising edge with sigmoid function, I need to use Levenberg-Marquardt iterative method (3.3.2), therefore I have to derive jacobian of this function for its parameters L, k, d and x_0 as

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f(x)}{\partial L} & \frac{\partial f(x)}{\partial k} & \frac{\partial f(x)}{\partial x_0} & \frac{\partial f(x)}{\partial d} \end{bmatrix} \tag{3.32}$$

, where

$$\begin{aligned}
\frac{\partial f(x)}{\partial L} &= \frac{1}{1 + e^{-k(x-x_0)}} \\
\frac{\partial f(x)}{\partial k} &= \frac{L(x - x_0)e^{-k(x-x_0)}}{-[1 + e^{-k(x-x_0)}]^2} \\
\frac{\partial f(x)}{\partial x_0} &= -\frac{Lke^{-k(x-x_0)}}{[1 + e^{-k(x-x_0)}]^2} \\
\frac{\partial f(x)}{\partial d} &= 1
\end{aligned} \tag{3.33}$$

Next, to get position of inflection point I have to derive sigmoid function twice and this second derivative compare to zero and extract variable x.

$$\frac{\partial^2 \frac{L}{1+e^{-k(x-x_0)}+d}}{\partial x^2} = \frac{-k^2 L e^{-k(x-x_0)}}{[1+e^{-k(x-x_0)}]^2} + \frac{2k^2 L e^{-2k(x-x_0)}}{[1+e^{-k(x-x_0)}]^3} = 0 \tag{3.34}$$

, so final result is

$$x = x_0 \tag{3.35}$$

which provides reason why to use logistic function in a form I used it - to get inflection point it is needed only estimate parameter x_0.

4 Solution

In this chapter I will describe the process of implementing chosen solution. In details I will discuss problems such as designing the marker, choosing suitable programming languages, building databases etc. Everything is designed in a way to minimize data traffic between parts of the application and increase self-sufficiency of system.

Inputs I have at the beginning are screen size in millimeters and device ID. No camera calibration is present.

4.1 Marker Detection

4.1.1 Marker design

In a case of marker design there are two main requirements - easiness of detection and width of application for as much devices as possible. Another requirement relates to 3D reconstruction - ideally about 20 detected points in known relative position spread around the screen. There are also limitations - the distance between screen and the nearest hardware button (or other active element) and a flatness of the desk panel of printer. Since most of the printers have flat front desk, I will ignore the second one.

I directly focused on contour-oriented detection, since the properties of environment (such as lightning conditions or camera position) are not defined (but at least bounded) - hence the template matching would not return maximum response in bad environmental conditions or with significant affine or projective transformation due to position of camera. Also, I did not want to set enormous number of rules and filters to extract feature points from the image. Therefore, I need high contrast marker which will be easy to describe. The final design has these advantages:

- it is high contrast against its background therefore easily detected as contour
- polygon shape has well defined vertices which do not depend on rotation of camera around Z axis. (this rotation is bounded to maximum of 30° in each direction)

meaning I can find all contours in image and then search for defined polygon. The reason why to use concave polygon is that it fits the corner of the screen which minimize the required space between screen and nearest active element and maximaze the distance between detected points. This shape is also easy to generate because the only number I need to know is the distance from the center of polygon to its vertex (polygon is symmetrical).

Final marker consists of four polygon shapes, each in a different corner of the screen.

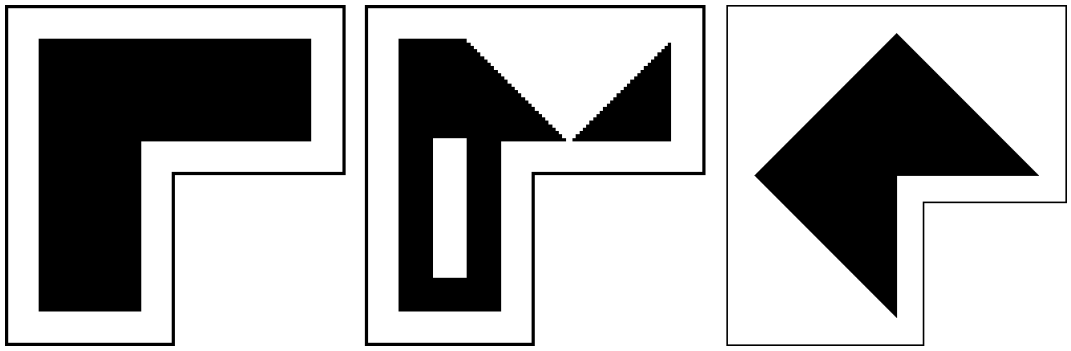

Fig. 4.1: Evolution of designed markers. From left - first design with problematic detection of corners due to rotation of camera around Z axis; second design as an experiment to get more feature points for 3D reconstruction; third and final design

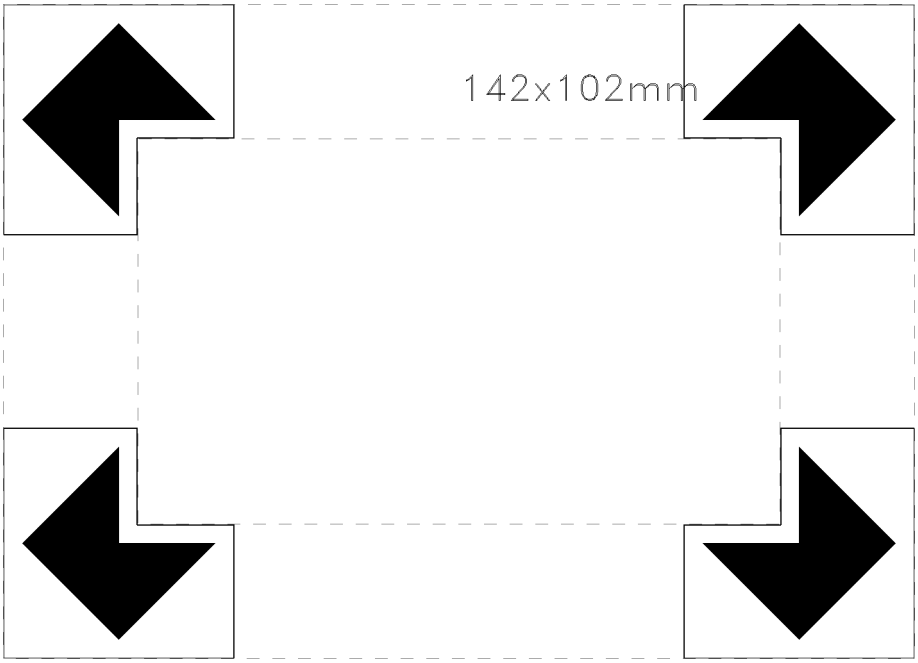

Fig. 4.2: Final version of marker frame. It consists of four corner shapes with the concave polygon shape.

4.1.2 Marker Detection

Along with marker design I developed a robust approach of detection. There are 3 main problems - find a threshold to binarize the image, find all four corner shapes of the marker and determine the position of each corner shape relative to the screen. The main idea behind finding corner shapes is based on finding all contours in the binary image and identify those belonging to marker.

Prefiltering contours

First step is to find all contours and reduce their number. I cut off too big and too small contours (by the number of their pixels) by following rules

$$\begin{aligned} n &> r/50 \\ n &< r/2 \end{aligned} \tag{4.1}$$

where n is number of pixels in contour and r is image perimeter. These rules were found by empirical testing with static range of camera-printer distance.

Due to presented polygon shape of marker's four corner parts and rotation boundaries I can claim that the top vertex will be always on the top of the contour, similarly the left, right and bottom vertices. Therefore, for every contour I determine the most left, right, up and down pixel. Then I test distribution of vertices of each contour, when the distance (in pixels) between two vertices must be less than $0.5 * n$ and more than $0.125 * n$ - so ideally should be close to one quarter of a number of contour's pixels.

Choosing corner shapes

For each contour which satisfied prefiltering I detect all five polygon vertices (four outer and one center vertex) by Douglas-Peucker algorithm (3.2.5). After that I connect neighboring vertex pixels of each contour with lines. Then I calculate the RMSE (eq. 3.16) of distance between every contour pixel and the corresponding line. For each corner shape I apply these conditions

- Concave part of contour must align to corner of the screen
- Contour must have 5 vertices

Then I order all contours satisfying these conditions by their RMSE and take first one for each corner shape. Final condition is that all chosen contours must have approximately the same values of RMSE for all polygon edges.

Threshold to binarize image

The biggest problem about marker detection was to determine right threshold to binarize the input image. Because the light conditions are unstable a single threshold was not sufficient to cover all possible outcomes. Therefore, I came up with iterative process of binarizing the image. Initially I binarize the image with the OTSU threshold, then try to find all four corner shapes of the marker. If there are not four contours, each with the RMSE's values satisfying the condition, the threshold is recalculated as $threshold_i = threshold_{OTSU} * coefficient_i$ and the detection is repeated. The coefficients were set empirically as $[1, 2, 0.3, 0.5, 3.5, 0.12, 6]$. After all

coefficients were used, the algorithm throws an exception that the marker cannot be detected.

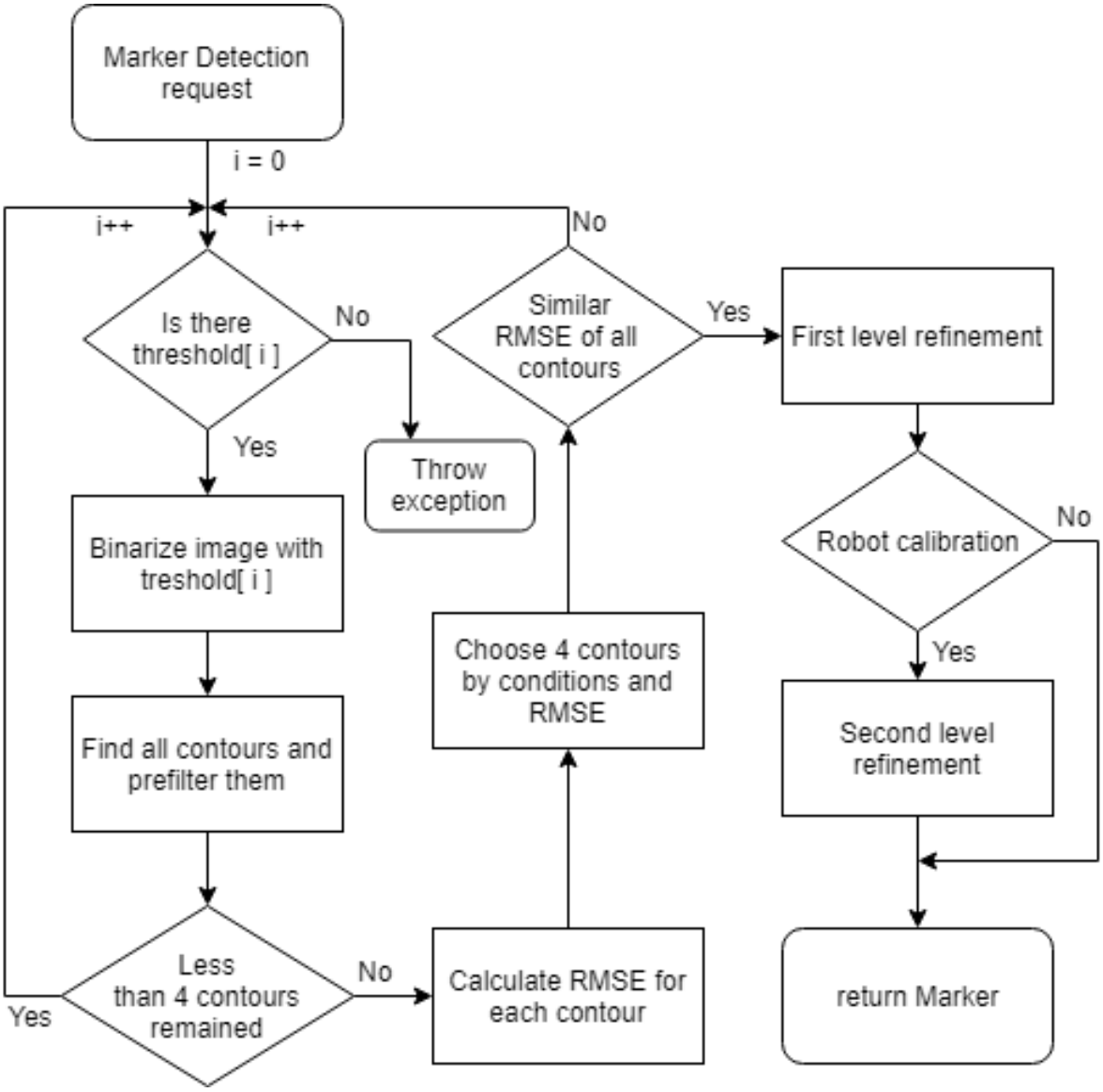

Fig. 4.3: Flow chart of marker detection.

This naive approach of getting the vertices as the most left/right/up/down pixel of the contour is used only to determine which four contours are part of the marker. For better accuracy I introduce two level refinement of detection.

First level of refinement

Now when I have contours representing four corner shapes and their polygon vertices I fit a line to every segment of contour bounded by neighboring vertices (all its

pixels) using least-squares method (3.1.6). After that I calculate intersections of all approximated lines. These intersections are then returned as a vertices of the marker.

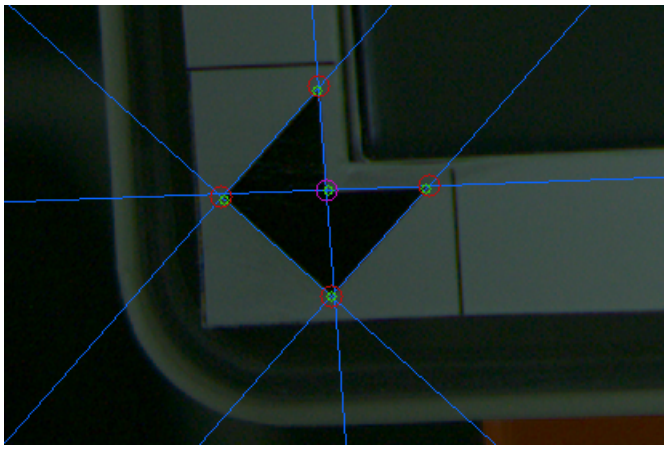

Fig. 4.4: Detected corner shape. Lines represent approximation by least squares, green circles represent vertices detected by Douglas-Peucker algorithm and red circles represent intersections of lines.

This approach is used for calibration of a camera view because it is fast and accurate enough.

Second level of refinement

To increase accuracy of marker detection, therefore also of 3D reconstruction, I use detection of inflection point. While first level of refinement still works with binary image, in this case I use the grayscale one. For every line between two neighboring vertices gotten by first level refinement I take its surrounding of 5 pixels in both perpendicular directions. This gives me n arrays of 11 pixels segments perpendicular to the line, where n is the length (in pixels) of the line. I approximate each segment with sigmoid function (3.1.5) using Levenberg-Marquardt (3.3.2) and by looking for the zero value of the second derivative of sigmoid function (3.4.2) I get the inflection point. Therefore, I get n inflection points per line, which again I approximate with line using least-squares method. After that it is the same - calculating intersections and returning them as vertices.

Before I chose the sigmoid function, I tested multiple functions to find the one which fits the data most - for example fitting polygon of 3-7 order, interpolation by spline, cubic interpolation etc.

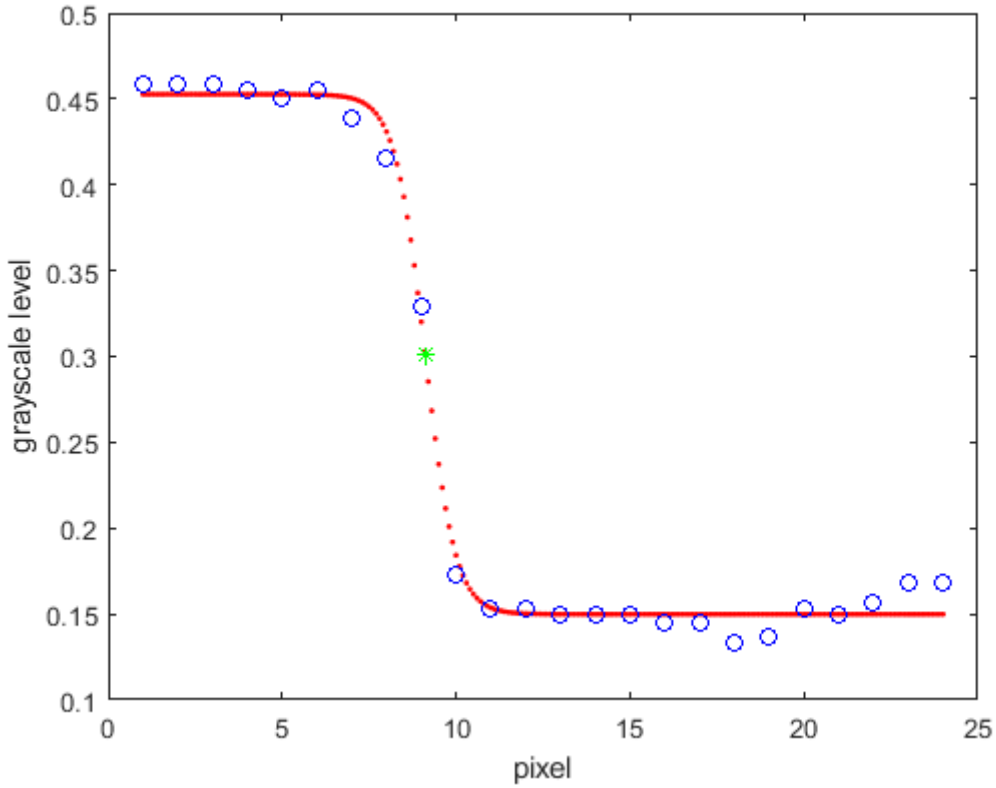

Fig. 4.5: Interpolated points with sigmoid function. Blue points are original points from image, red points are interpolated by sigmoid function and green star represents inflection point.

4.2 Calibration of Camera View

Thanks to marker detection I got the information about position of vertices of each corner shape. Since I generate these markers, I also have the information about the offset of corner shapes from the edge of the screen (meaning the edge of the marker frame). For each edge of the screen I select two corner shapes aligned to the edge. From those two shapes I extract all vertices which form line parallel to the screen edge. Then I fit a line to those points for each edge. After that I calculate intersections of these lines. Now I should have more accurate position of center vertices of corner shapes. The problem is that all corner shapes are shifted from real corners of the printer screen to increase robustness of contour detection. This offset is known since I generate these markers, therefore I am able to remove this offset. First, I get model of intersections in screen coordinate system by shifting screen corners by offset (screen model is known thanks to screen size). After that I calculate perspective transformation from known intersections model (in millimeters) and

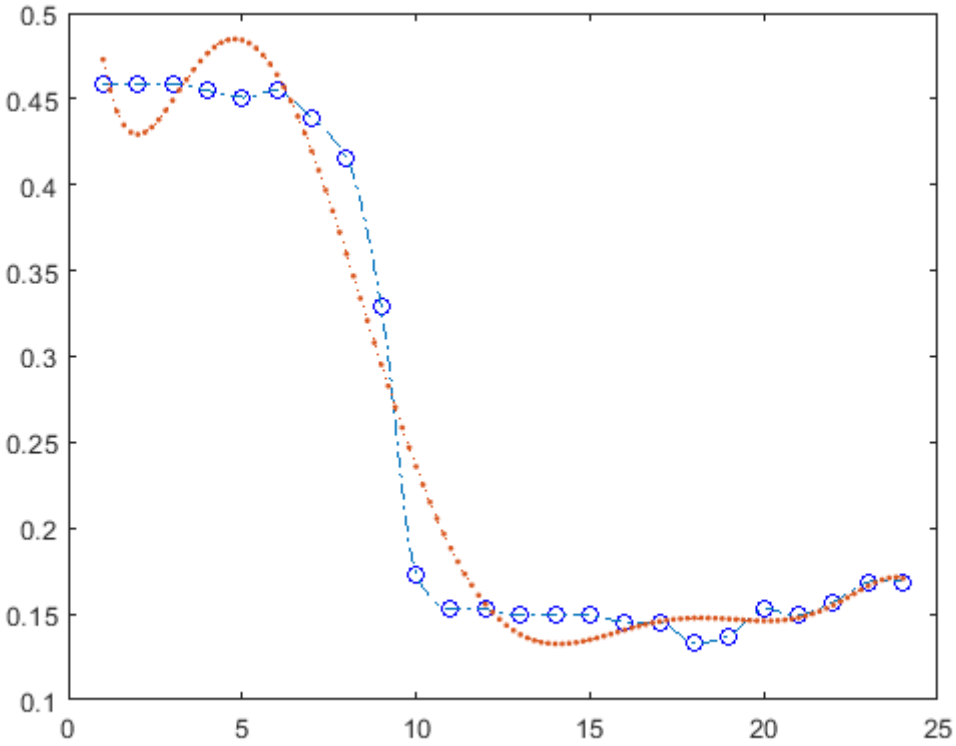

Fig. 4.6: Examples of tested interpolation options - blue points are spline interpolation, orange ones are polygon fitting of 7th order

actual intersections (in pixels). Finally, using this transformation I transform screen corners model to coordinate system of the image thus I get true corners of screen region in the image. Those four points are then returned as a vertices of a screen region.

4.3 Calibration of Robot Arm

4.3.1 Robot Arm Detection

To calibrate the robot arm with one camera view I need to have following information

- detected robot arm in the picture
- model of the robot arm
- detected printer screen in the picture
- model of the printer screen

I already solved the two last items of the list by marker detecting and knowing its model. To get the information about the position of the arm in the image and its

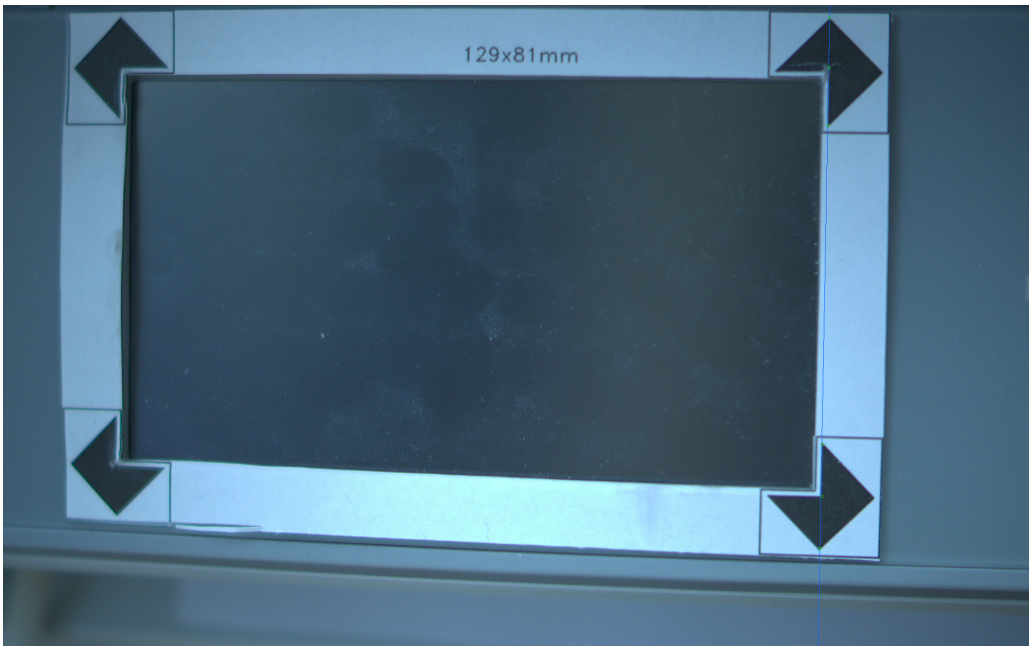

Fig. 4.7: Selected points for screen edge and line which fits these points.

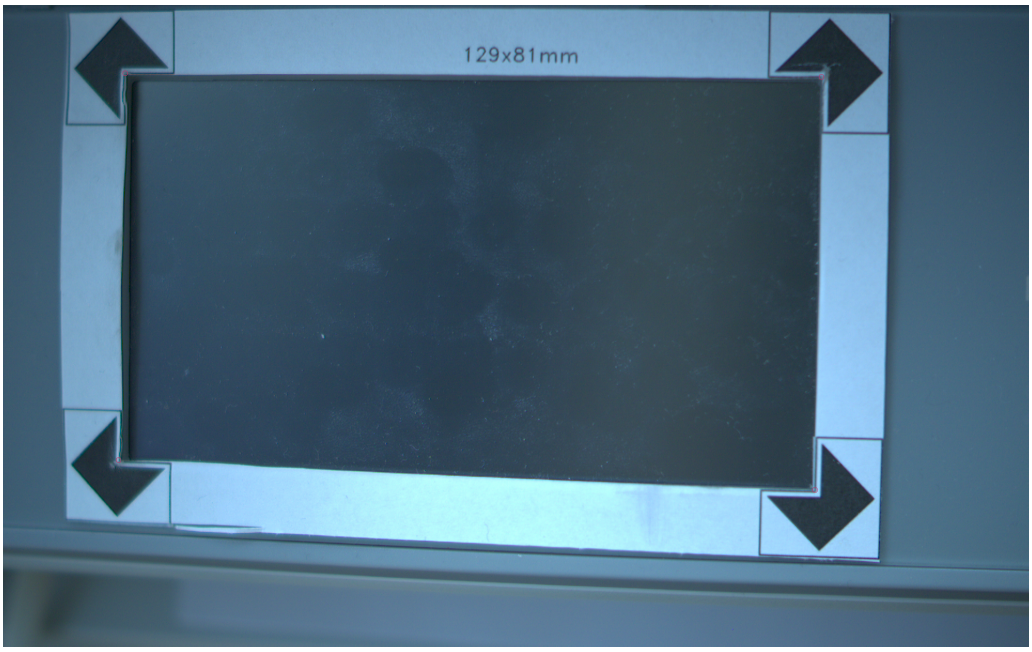

Fig. 4.8: True corners of screen region in the image.

model I needed to put a marker on the arm too. Because the issue of area around the robot is not that crucial as in case of the printer screen I decided to put a chessboard of known model in front of the robot arm. There was also a possibility to put the chessboard (or other known structure) directly onto the arm, but I chose

the first option for these reasons

- it is not possible to get robot's tripod really close to printer (always some space between them)
- camera needs to see both the marker and the chessboard
- it is desirable to have most of the image filled with the printer screen (for better resolution for image processing)
- the initial position of the arm is the arm leaning back

Calibration of the robot arm is supposed to be used mostly before the arm is turned on so having known pattern directly on the arm would lead to problem with low resolution of the printer screen region in the image. Therefore, I designed a chessboard holder with a known relative position to the origin of robot coordinate system and then printed it using 3D printer.

Fig. 4.9: 3D printed chessboard holder installed on the robot arms.

Using the chessboard, I detect all inner corners using OpenCV algorithms.

4.3.2 3D Reconstruction

Now I use Gauss-Newton iterative method to determine camera position in coordinate system of printer screen and robot arm. So, I use camera as a middle point to get transformation from robot to printer screen. The algorithm is the same for both cases, based on 3.4. The only parameters which differ are inputs - specifically model of pattern and detected pixels of pattern. All properties I need to start iteration are

- Detected pattern (in pixels)
- Model of pattern (in millimeters)
- Camera intrinsic matrix
- Initial Estimation

Model of pattern must be defined in coordinate system of printer screen/robot arm. Camera intrinsic parameters are obtained by camera calibration which had to be introduced (more in sec. 4.4). Initial estimation is the same for both cases and was set to $[0, 0, 800, 0, 0, 0]$ for parameter vector $\begin{bmatrix} t_x & t_y & t_z & \alpha & \beta & \gamma \end{bmatrix}$ (by 3.25), because the camera should be about 80cm above the screen with zero rotations and zero translations in x and y axis. Then the algorithm goes:

1. Inputs: model, detected pattern, intrinsic matrix, initial estimation
   Initiate: params[0] = initial estimation
2. While (L2 norm of parameters from previous iteration $i-1$ and this iteration i is bigger than $Norm_{min}$)
   AND
   (iteration i is smaller than $iteration_{max}$)
   do :
   (a) Get jacobian **J** by 3.28
   (b) Get Δ by 3.23
   (c) Save previous parameters and calculate new ones by 3.21
   (d) Calculate L2 norm of parameters from previous and this iteration
3. Return parameters

To get jacobian **J** we use model points as vector **X** (3.30), to get error ϵ we use detected points as vector **x** (3.19) and intrinsic matrix as matrix **K**.

When I get both estimated parameter vectors for printer screen and robot arm, I calculate transformation from printer coordinate system to robot coordinate system as

$$\mathbf{T_{p2r}} = \mathbf{T_{p2c}}{\mathbf{T_{r2c}}}^{-1} \tag{4.2}$$

where T_{p2r} is transform matrix from printer screen to robot coordinate system, T_{p2c} is transform matrix from printer screen to camera and T_{r2c} is transform matrix from robot to camera. Transform matrix from printer/robot to camera are gotten from parameters by 3.3.

There are two minor problems - first is that chessboard model and detected chessboard points are shifted from the origin of robot arm by proportions of chessboard holder. Secondly, not all screens are in plane with front panel of the printer where the marker is attached. Some of them are recessed to front panel. Both the recessing of screen and offset of chessboard holder are known therefore simple adjustment of translation vector of both parameter vectors is needed before $\mathbf{T_{p2c}}$ and $\mathbf{T_{r2c}}$ are calculated.

4.4 Camera Calibration

During the testing of automated calibration of robot arm occurred a problem with uncalibrated cameras. Since the camera had been used only for detecting printer screen, its calibration was not an issue for the whole system. But it turned out that it is necessary for accurate robot arm calibration. To get these parameters, I wrote a calibration method which uses 3rd party openCV tools. First, set of images containing calibration chessboard of known model is needed. Then using $findChessboardCorners$ method I find chessboard corners for each image. The method uses adaptive thresholding to binarize the image and morphological operations to separate chessboard squares into quadrilaterals by contour searching [4][5]. After that subpixel refinement is used by $cornerSubPix$ method ([6]). Finally, $CalibrateCamera$ method is used to get intrinsic parameters and distortion coefficients by Levenberg-Marquardt iterative process. (3.3.2)

4.5 Light

Part of the solution is also a problem with light conditions. Marker detection and chessboard detection are robust enough to detect their object of interest in various light conditions but the biggest problem with detection is when there is lack of light, especially during the night or in dark rooms. Because the robot system operates in offices where there is no possibility to control light sources, it is convenient to attach the light source directly to the system. Therefore, I designed a light which can be put on the camera and is 3D printed. The light head is designed in the way to not directly radiate to the screen/robot arm to reduce glints but rather to use indirect light of reflections from the inside of the head.

Around an inner edge of the lamp there is set a LED strip. To prevent light to radiate directly onto the object (printer/robot) I put a raised edge with a cone shape which reflects all the light up into the lamp head and then out. This helps to diffuse light more.

LED stripe is used as light source and inside of lamp head is painted white to maximize light reflection.

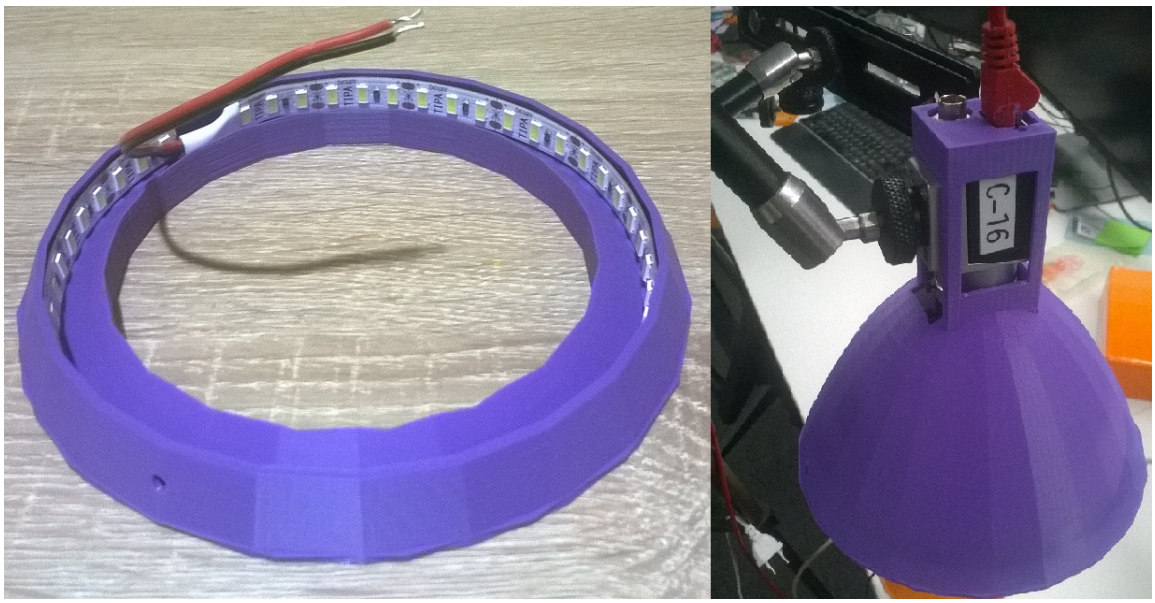

Fig. 4.10: On the left - designed light from inside with a LED strip. On the right - light with camera.

5 Implementation

From the implementation point of view, the whole solution is divided into three parts - C#, F# and database project of .NET framework. It uses open source OpenCV library - more specifically its wrapper OpenCvSharp since the library is not available for C# (or F#) - and Microsoft Visual Studio environment.

C# project takes care of marker detection, detection refinement, marker generating and calibration of both the camera view and the robot arm and is organized following these functionalities into concrete folders. This project also refers to other two projects to handle database objects and to use iterative methods written in F#.

Database project is implemented using Entity Framework and Code First approach. Instead of creating database model first and after that define all classes, this approach allows to define classes first and the database model is created based on these classes.

F# project includes implementations of all iterative methods - Gauss Newton for 3D reconstruction, Levenberg-Marquardt for sigmoid interpolation and handling matrix transformations. The reason why to use F# to implement these parts is that the way how C# handles matrices and their calculations is really heavy to compare, for example, with Matlab. So, I tried to find a .NET compatible programming language which is better in this task. Other languages like Python or R are also good to handle matrices but their compatibility with .NET framework is not that straightforward as it is with F#. For comparison a simple code sample of assigning values to submatrix at fig. 5.1

Listing 5.1: Comparison of C# and F# handling matrices.

```
// C#
var m1 = Matrix.Build.DenseOfArray(
      new[,] {{1.0, 2.0, 1.0},
              {2.0, 3.0, 1.0},
              {3.0, 5.0, 0.0}}
    );
var m2 = Matrix.Build.Random(3,3);
m2.SubMatrix(1,2,1,2).AssignTo(m1[Range(1,2)][Range(1,2)]);
// F#
let m1 = matrix [[1.0; 2.0; 1.0]
                 [2.0; 3.0; 1.0]
                 [3.0; 5.0; 0.0]]
let m2 = DenseMatrix.random<float> 3 3
m2.[1..2][1..2] <- m1.[1..2][1..2]
```

As reader can see the code written in F# is much more readable. Also, F# as functional language compiles all commands in order as they are written in code file, meaning it is possible to call only functions and variables which are defined earlier in the code (above the caller). This corresponds to a way how iterative methods are calculated. F# also implements all C# libraries so in case of lack of suitable F# function I can call C# equivalent.

In following sections I will describe main classes.

5.1 Marker Generator

MarkerGenerator class provides functions needed for marker generating. The only public method is *GeneratePdf* which returns FileStream of newly created .pdf file containing marker frame. The file is created in the user Temp folder and is prepared to be directly printed on A4 paper format. User then has to cut the frame with scissors/cutter using dashed lines (can be seen in fig. 4.2). Returned FileStream is then used by superior web application which communicates with the user.

Inputs are *screenSize* and *minDistanceAroundScreen*. To avoid storing the whole marker model in database I use only distance between screen edge and the nearest active element - called *minDistanceAroundScreen* - for every device. From this value I create MarkerTemplate object which generate all marker model points by these rules

- throw exception if *minDistanceAroundScreen* is less then 3mm, otherwise
- set shape offset to min(1,*minDistanceAroundScreen*/10)
- rest of the space fill with the corner shapes (fig. 5.1)

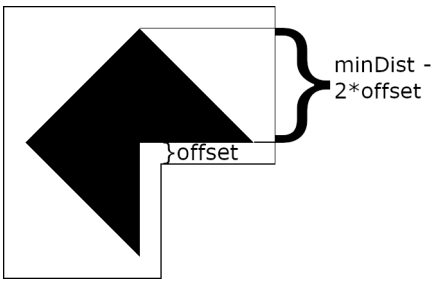

Fig. 5.1: MarkerTemplate distribution of *minDistanceAroundScreen*.

5.2 Marker Detector

MarkerDetector class follows flow chart from fig. 4.3. Its inputs are MarkerTemplate mentioned earlier and OpenCV class Mat representing camera image. Simplify flow chart is presented in fig. 5.2. After finding all contours and prefiltering them, ContourForEval is created from every remaining contour. This class is responsible for approximating contour with polygon using ApproxPolyDP method from openCV library, fitting lines to edges of this polygon and calculating RMSE of every polygon edge. In other words, it prepares everything for further decision if the particular contour is corner shape or not. Then method FindCornerShapes chooses from List of ContourForEval four contours which represent four corner shapes and returns List of CornerShape classes. CornerShape class reorganizes ContourForEval for further use by CameraCalibrator and RobotCalibrator. It also assigns location to each contour.

The MarkerDetector class has one public method Detect with one bool argument which returns Marker class. This class contains List of four CornerShape classes representing pixel points of detected marker and MarkerTemplate class which represents marker model points in milimeters. The argument serves as a flag for second level refinement (4.1.2).

5.3 Calibrators

CameraCalibrator class takes care of calibration of camera view. It has one public method Calibrate which has 3 parameters

- deviceModelId - ID of device
- screenSize
- img - input image

. Then the algorithm goes like this

1. Argument deviceModelId is used to get Device object from database. This object contains minDistanceAroundScreen property and is able to return MarkerTemplate with help of this property by calling GetMarkerTemplate method.
2. Then marker is detected in the input image by MarkerDetector and its method Detect(), which takes MarkerTemplate as an argument.
3. For each screen edge is called GetSidePoints() method and returned points are approximated with line by method Cv2.FitLine() from OpenCV library
4. Intersections of lines are calculated by CalculateIntersectionsOfLines() method
5. Intersections are shifted by offset by calling AdjustIntersectionsByMarkerOffset() method
6. Shifted points are returned as result

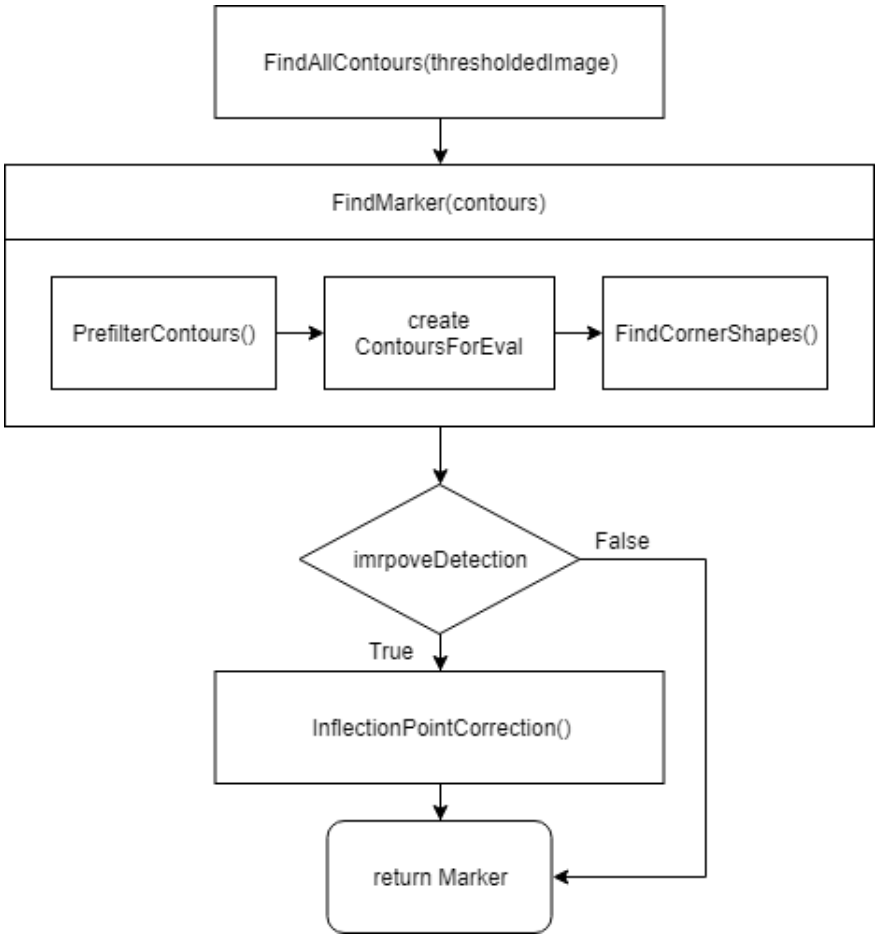

Fig. 5.2: Simplified flow chart of MarkerDetector.Detect() method.

Part of this algorithm is also validation of detected corners. It only checks if bottom points are lower than upper points and left points are more left then right points.

RobotCalibrator class contains methods to solve calibration of robot arm. Its one public method Calibrate() takes 5 arguments

- deviceModelId
- screenSize
- robotId
- camId
- img

Because of higher accuracy demands, I had to introduce two objects - Robot and Camera. Robot class encapsulates properties belonging to every robot arm, which are ChessboardModel (containing size of a square and number of squares in 2 dimensions) and TranslationChessboardToArm (representing translation vector from chessboard holder to origin of robot coordinate system). Camera class contains intrinsic parameters - IntrinsicMatrix and DistortionCoefficients. It also contains

static method GetIntrinsicParameters() which takes List of images, chessboard model and its size and returns camera matrix and distortion coefficients using OpenCV methods. Both the Camera and the Robot class refer to real hardware by Id property. Then the program goes like

1. Get Device, Camera and Robot instance from database by their Id
2. Get transformation parameters for screen by calling ScreenToCamera() method
3. Get transformation parameters for robot by calling RobotToCamera() method
4. Get final transform matrix robot-> printer by calling F# method transform_rbt_to_printer (following eq. 4.2)

Both ScreenToCamera() and RobotToCamera() detect their pattern (marker/chessboard) and prepare its model. Then they call GaussNewtonFs() method which handles F# library with iterative Gauss Newton method.

GaussNewtonFs() method takes care of the iteration process, meaning it looks after conditions which stop the iteration and passes data between iterations. It also calls F# method GaussNewton.one_iter() which provides all calculations needed to proceed one iteration of the algorithm and returns estimated parameters.

5.4 Database model

The database is designed to store only necessary data which cannot be simplified. That means the database includes 5 tables - Cameras, Robots, Devices, MatrixElements and ChessboardModels. Database diagram in fig. 5.3 shows relations between tables and their content.

I will only describe properties which could be confusing. MinDistanceAroundScreen in Devices table is used to create MarkerTemplate class (representing marker model). NumberOfInnerPoints in ChessboardModels table is 2-element vector which represents number of detected points by OpenCV method in 2 dimensions (it is basically number of squares minus one). MatrixElements table consists of columns X and Y, which determine the position in matrix, and column Value representing numerical value on that position. Last thing which might not be clear is meaning of DeviceModelId. Unlike DeviceId, which is primary key for Devices table and is referring to specific record, DeviceModelId serves as a pointer to real piece of hardware, for instance printer model (e.g. Konica Minolta C364). So when superior application wants to calibrate specific printer, this property is used to get right MarkerTemplate or ChessboardModel.

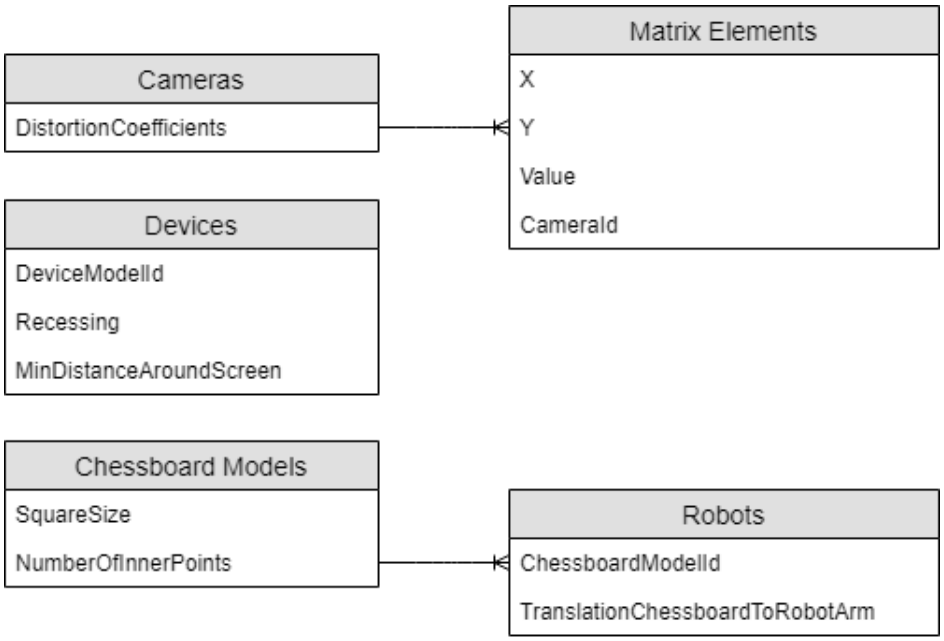

Fig. 5.3: Database diagram.

5.5 F# Library

F# project consists of two files divided into three modules. First module is called GaussNewton and includes functionality to proceed one iteration of Gauss Newton iterative method. It calculates individual components of rotation matrix, its derivative, projection matrix, processes 3D and 2D coordinates of model points (in mm) and detected points (in pixels) respectively to homogeneous coordinates, calculates back-projection error, gets jacobian, calculates delta and finally new values of parameter vector which returns as result.

Second module is matrixTransform module and its only usage is to get final transform matrix from robot to printer screen from partial results of transformations to camera.

Third module - Levenberg-Marquardt - provides methods for processing one iteration of Levenberg-Marquardt iterative method which here interpolates edge of detected corner shape with sigmoid function. Similarly to Gauss-Newton, it first calculates outputs for given parameters of sigmoid function (from previous iteration), then it calculates error ϵ (pixel values minus sigmoid outputs), then gets jacobian, delta and new parameter vector.

5.6 Matlab

Most of presented functionalities were first tested in Matlab for its vision-and-matrix-friendly environment. Specifically: one camera pose estimation, second level refinement by detection of inflection point, marker and chessboard generating, detection of laser cross and calibration of camera intrinsic parameters. I will not describe these files in details but I will include them in the annexes with commentary.

detection output	note	number [-]
Detected	true positive	379
	false positive	3
	corrupted	10
	robot fault	151
Undetected	true negative	117
	dark	311
	false negative	16

Tab. 6.1: Results of marker detection. In *note* column there is description of detection or its cause.

6 Tests and Results

At the beginning I would like to say that I am not able to calibrate robot arm accurate enough that it is able to tap on the screen precisely or safely. Most of the tests of which you will read results were made to figure the way how to push the precision further and what is necessary to change in the system. At the end I will summarize the results and introduce some of my ideas on what should be the next move.

6.1 Marker Detection

In terms of marker detection there is one main goal - detect vertices of four corner shapes as precise as possible. As part of camera view calibration, the robotic system has been using marker detection for more than 7 months. I took images taken due to camera view calibration request in two days on 4 devices and in this way, I collected 987 images of printer screen with marker frame. I went through the set by marker detection algorithm and tested the set for true/false positive/negative detections. Detailed results are presented in tab. 6.1.

True positive detections are those which were able to detect all 4 corner shapes with a sufficient accuracy usable for camera view calibration. False positive detections detected wrong contour as a corner shape, corrupted detections detected all corner shapes but with insufficient accuracy (mainly due to wrong threshold) and class "robot fault" encapsulates cases when robot arm slightly covered one of corner shapes which led to uncomplete detection of that corner shape. Examples of these situations are in figures 6.1, 6.2, 6.3 and 6.4.

For undetected markers there are following categories - true negative detection represents images where one of corner shapes is missing (due to camera shift, robot

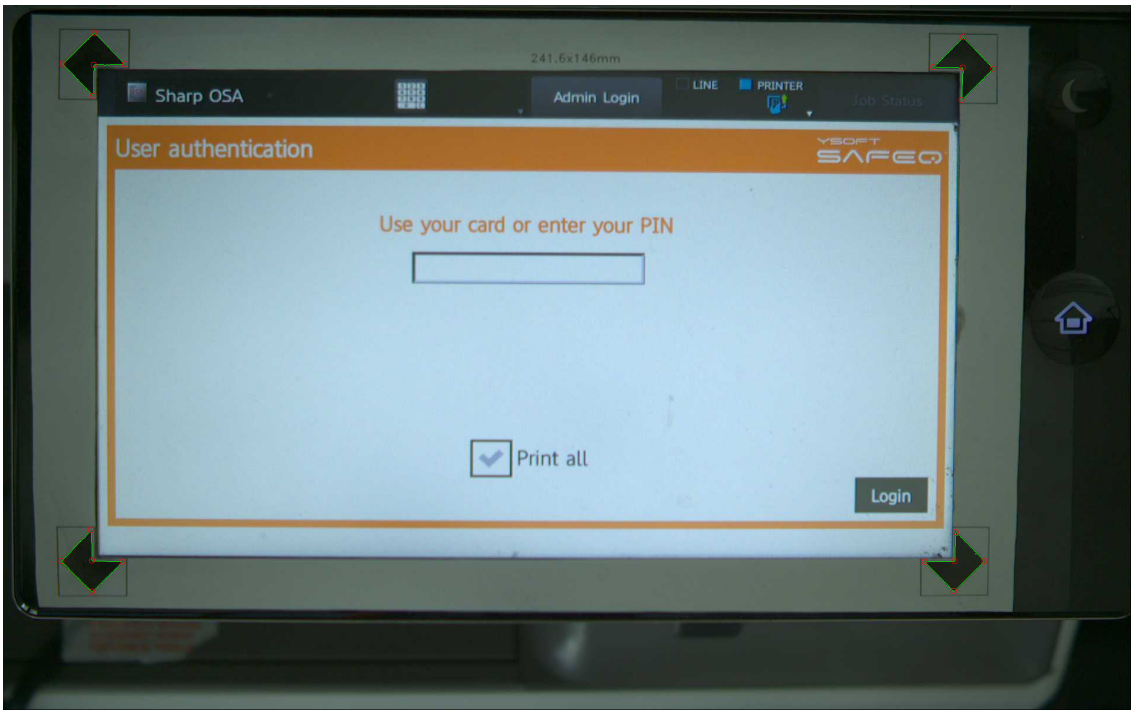

Fig. 6.1: True positive marker detection.

Fig. 6.2: False positive marker detection.

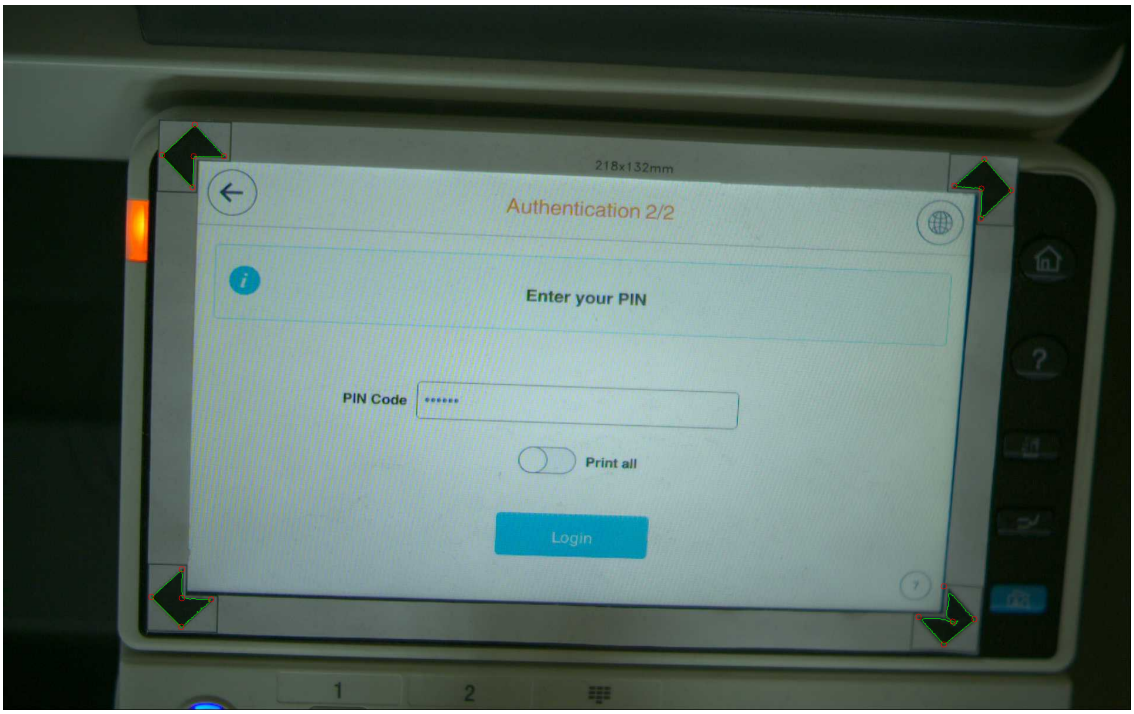

Fig. 6.3: Corrupted marker detection.

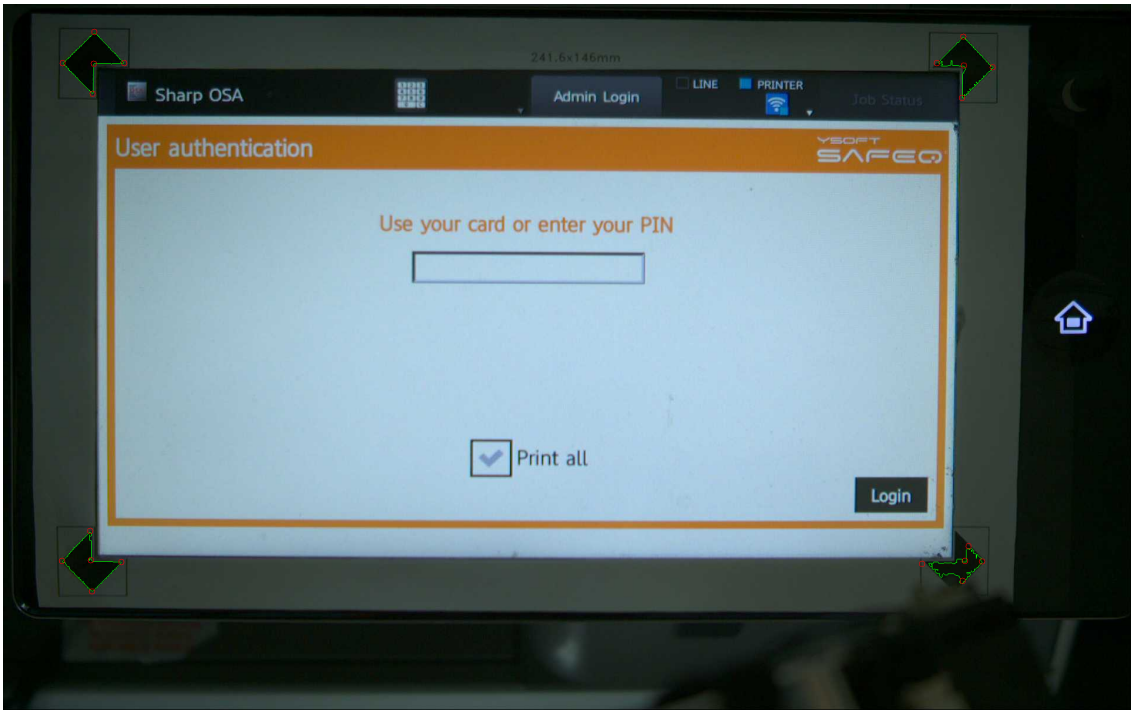

Fig. 6.4: Robot arm covering part of corner shape.

		True marker presence	
		positive	**negative**
Detected presence	**TRUE**	379	117
	FALSE	13	16

Tab. 6.2: Confusion matrix of marker detection.

printer vendor	**calibrated**	**validation failed**
Brother	29	0
Km	223	2
Oki	58	0
Sharp	230	1

Tab. 6.3: Camera view calibration results for 4 vendors.

arm etc.), "dark" class contains images with bad light conditions, false negative contains images where algorithm was supposed to detect marker frame. The reason why to create "dark" class is that this problem should solve designed light (4.5) which right now is waiting for implementation of 5V switch so it is able to flash only when there is calibration request (it is not used right now because it would have to be switched on non-stop).

Considering categories explained above I set a confusion matrix by skipping all "dark" undetected markers, also skipping all "robot fault" detected markers since it is an operator responsibility to set camera in a way that it can see marker (even most of the detections provide accuracy high enough to proceed camera view). Next I merged false positive and corrupted categories since their outcome is mostly the same from calibration point of view, and finally true positive, true negative and false negative stay unchanged. Final confusion matrix can be seen in tab. 6.2.

6.2 Camera View Calibration

Camera view calibration is used in two situations - firstly when the whole system is set in a new place in the front of a device. Then it is necessary to calibrate both camera view and robot arm. Second situation is when image processing algorithms are not able to recognize screen content therefore the system is not able to continue with tests. One of common reasons is slight displacement of camera because someone in the office accidentally push the tripod away. Then it is convenient to try to recalibrate camera view. For testing camera view calibration, I used the same data set as in case of testing marker detection, but only those images where I was able to detect marker. So final set consists of 543 images. Results can be seen in tab. 6.3.

The algorithm is able to calibrate camera view for majority of marker detection. Validation described in sec. 5.3 caught all marker misdetections from tab. 6.1. It also calibrated all corrupted detections and "robot fault" detections. Superior algorithm then decides if the calibration is accurate enough or it needs new one. Example of rejected marker detection by calibration validation is shown in fig. 6.5.

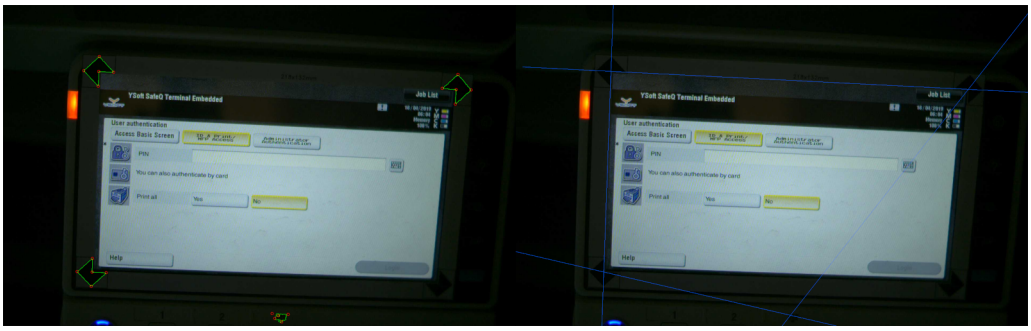

Fig. 6.5: Camera calibration did not validate marker detection. On the left there is detected marker, on the right there are fitted lines.

6.3 Robot Arm Calibration

The problem with testing robot arm calibration is that I am not able to measure separate parts of calibration algorithm (marker and chessboard reconstruction) due to unknown position of camera and measured object (marker/chessboard). At least not in reasonable time or prize. Therefore, I have to test it as a whole using precisely printed test cases (marker with chessboard in known relative rotation and translation) since tests with real hardware should follow after theoretical validation. One of these test cases can be seen in fig. 6.6. For testing robot calibration, I generated several test cases simulating different settings of screen-robot (meaning marker-chessboard). It consists of 5 settings - chessboard translated by 20, 30, 40, 50 and 60mm in x and y axis - and 2 light conditions - with and without light, meaning 10 test cases in total.

6.3.1 Marker Reconstruction

One of the important precision indicators is back-projection error. It can provide information how well estimated parameters really represent true position of camera in space (relative to screen/chessboard). If the back-projection error is big, then the iterative process got stuck in local minimum far from the global minimum. It depends on detection precision as well as on right setting of the iteration parameters

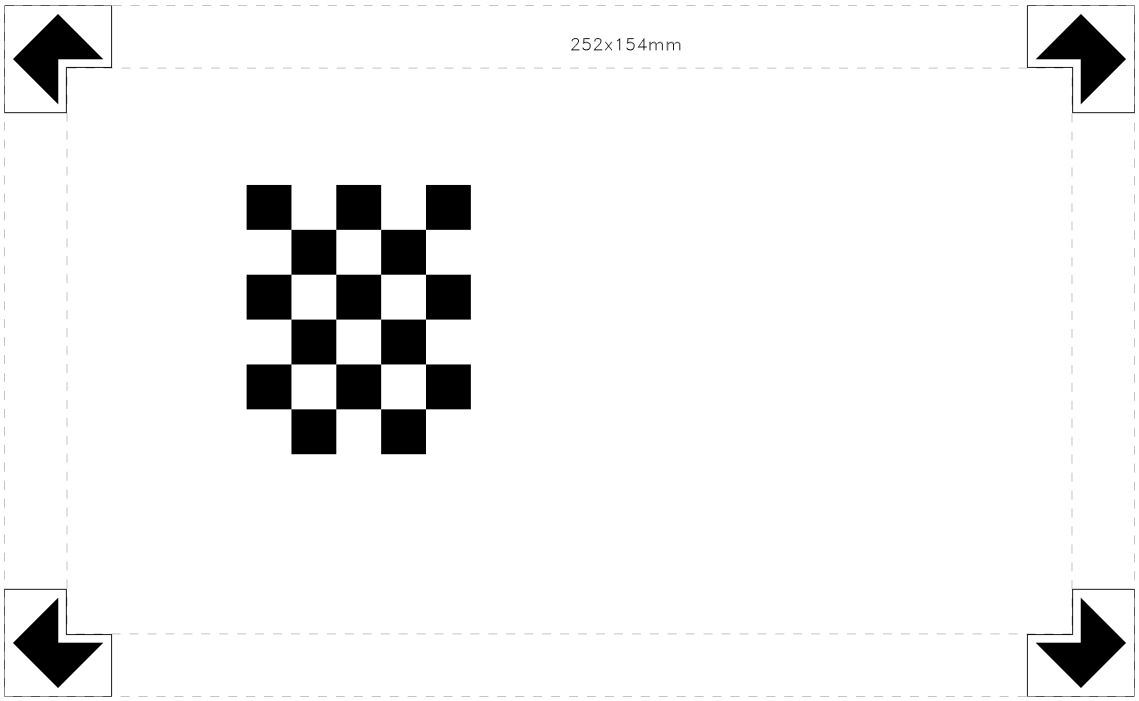

Fig. 6.6: Test case for chessboard translated by 5cm in X and Y axis.

(maximum number of iterations, termination conditions). As I cannot determine relative position camera-screen or camera-robot, I will use back-projection error to measure quality of 3D reconstruction.

Frame vs 4 separate corner shapes

At the beginning there was an idea to mark the screen not with the whole frame, but only with four pieces of corner shapes. The reason why to use four separate pieces was that the user could just once print a list of paper with tens of corner shapes and then use and change them one by one. The problem occurred during back-projection testing when the positioning of four separate pieces turned out to be less precise as it is set by user. Usage of a frame improves the result because it keeps distances between shapes unchanged from the model. An example of back-projection error results for these two cases is in tab. 6.4.

As reader could expect, the frame has got smaller error therefore it is more suitable for robot calibration due to its higher precision.

Second level refinement

Second test is focused on precision of second level refinement. I proceeded all 10 test cases through reconstruction with and without refinement and as expected, the

point	Back-projection error [pixel]	
	4 corner shapes	frame
1	6.63	5.83
2	1.38	1.33
3	8.49	6.01
4	5.67	3.31
5	4.97	4.58
6	7.08	6.35
7	5.46	5.49
8	2.90	4.60
9	8.51	3.81
10	2.09	5.39
11	2.96	0.49
12	5.83	5.83
13	3.52	2.76
14	4.89	4.25
15	7.70	1.08
16	6.11	2.93
17	8.48	0.82
18	8.46	4.17
19	4.76	1.80
20	6.04	4.78
Average	5.59	3.78

Tab. 6.4: Results of back-projection errors of all 20 marker points for 4$cornershapes$ and $frame$ test case and their average.

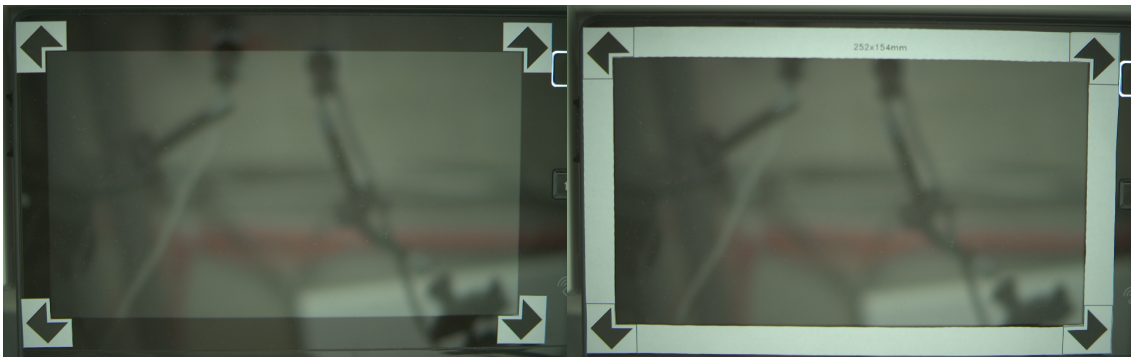

Fig. 6.7: Image of marker consisting of four pieces (left) and of one solid frame (right).

second level refinement got smaller back-projection error in most cases (8 out of 10). The reason why in minority of cases is result better without second level refinement, is that it can happen that the thresholding by one of the given thresholds is actually really close to ideal threshold and the light conditions are homogeneous in whole image. Results can be seen in tab. 6.5.

Light

Although the main purpose of designing own light was to enable calibration during night and bad light conditions, I ran a test to figure if the added light increase precision of reconstruction in some extent. Tab. 6.6 shows results of reconstruction with second level refinement for images with and without additional light. As reader can see the difference is more random than systematic. It corresponds to nature of second level refinement algorithm because more light makes the edge between black marker and white background steeper but it does not move the inflection point or even does not make his detection easier in any way.

6.3.2 Calibration

Tab. 6.7 shows results of whole robot calibration application and their comparison with true value, using intrinsic parameters gotten from all 73 images of calibration chessboard (sec. 6.4.1) and basic model of higher order distortion (3.1.7). All 10 test cases were used. Reader can see that the results show insufficient accuracy to use this calibration alone with the real system. Maximal sample standard deviation for translation is almost 7mm for x axis and for rotation it is almost 0,01rad for rotation around y axis (β). Maximal error for translation is for test 6, x axis - 13.4mm. For rotation the maximum error is also for test 6, β - 0.018rad (1.05°). Most of the tests

test number	Average Back-projection error [pixel]	
	Without Refinement	With Refinement
1	7.84	2.08
2	1.49	1.95
3	1.80	1.86
4	2.39	1.90
5	2.16	1.98
6	2.83	2.13
7	2.68	2.10
8	2.90	2.19
9	2.31	2.05
10	2.23	2.09

Tab. 6.5: Results of average back-projection error of 20 detected marker points for 10 test cases, each with and without second level refinement.

test	Average Back-projection error [pixel]	
	Light	No Light
1	2.08	1.95
2	1.86	1.90
3	1.98	2.13
4	2.10	2.19
5	2.05	2.09

Tab. 6.6: Results of average back-projection error of 20 detected marker points for 10 test cases sorted by light conditions.

test	t_x [mm]	t_y [mm]	t_z [mm]	$\alpha[rad]$	$\beta[rad]$	$\gamma[rad]$
true value	-20	-20	0	0	0	0
1	-24.91	-29.53	-0.77	-0.01349	0.00636	0
2	-20.65	-29.7	-0.26	-0.01346	0.00056	-0.00028
true value	-30	-30	0	0	0	0
3	-21.61	-32.36	-0.62	-0.00264	-0.01122	-0.0004
4	-19.99	-32.07	-0.79	-0.00225	-0.01342	-0.00052
true value	-40	-40	0	0	0	0
5	-31.14	-34.63	-1.65	0.00774	-0.01221	-0.00059
6	-26.6	-34.54	-1.37	0.00789	-0.01847	-0.00094
true value	-50	-50	0	0	0	0
7	-47.23	-54.27	-2.72	-0.00526	-0.00327	0.00037
8	-47.84	-54.25	-3.01	-0.00524	-0.0024	-0.00011
true value	-60	-60	0	0	0	0
9	-59.09	-55.97	-4.22	0.0058	-0.00101	0.00018
10	-57.33	-57.19	-4.29	0.0041	-0.00342	0.00011
σ	6.88	5.60	2.43	0.00776	0.00931	0.00044

Tab. 6.7: Robot calibration test for all 10 test cases.

in this section were performed to find the cause of this inaccuracy and to answer the question what steps must be done to get the system into an autocalibrating state.

6.3.3 Principal point and focal length shift

One of the parameters of robot calibration which could be prone to be crucial is intrinsic matrix of camera. Therefore, I tested one test case for different principal point and focal length shifts, separately. Results are presented in tab. 6.8. As the table shows, parameter estimation does not differ that much if the principal point coordinates C_x and C_y are changed, even for shift of ± 15 pixels. On the other hand, the shift of focal length is significant - for f_x shift of ± 15 pixels it differs by more than 16mm in x axis translation and 8mm in y axis translation. The difference is the same for f_y shift. For shift of ± 15 pixels, the difference is also biggest in translation along x and y axis - 15mm and 8mm respectively. Rotation difference is biggest for f_y shift by 15 pixels - almost 1.5° (0.025rad) for β.

	shift [pixel]	t_x **[mm]**	t_y **[mm]**	t_z **[mm]**	$\alpha[rad]$	$\beta[rad]$	$\gamma[rad]$
origin	0	-24.91	-29.53	-0.77	-0.01349	0.00636	0
C_x	-15	-24.48	-30.81	-0.57	-0.01526	0.00574	-0.00005
	-10	-24.63	-30.38	-0.64	-0.01467	0.00596	-0.00004
	-5	-24.78	-29.95	-0.71	-0.01407	0.00617	-0.00002
	5	-25.04	-29.11	-0.84	-0.0129	0.00655	0.00002
	10	-25.17	-28.69	-0.9	-0.01232	0.00672	0.00004
	15	-25.28	-28.28	-0.96	-0.01175	0.00689	0.00006
C_y	-15	-23.73	-29.26	-0.51	-0.01312	0.00471	0.00001
	-10	-24.12	-29.35	-0.59	-0.01324	0.00525	0.00000
	-5	-24.51	-29.44	-0.68	-0.01336	0.0058	0.00000
	5	-25.32	-29.62	-0.86	-0.01361	0.00694	0.00000
	10	-25.74	-29.71	-0.96	-0.01373	0.00752	0.00000
	15	-26.16	-29.8	-1.05	-0.01385	0.00811	0.00000
f_x	-15	-38.49	-37.79	-2.75	-0.02493	0.02534	-0.0007
	-10	-34.36	-34.77	-2.22	-0.02074	0.01957	-0.00042
	-5	-29.82	-31.99	-1.56	-0.01689	0.01323	-0.00018
	5	-19.72	-27.43	0.14	-0.01059	-0.00091	0.00013
	10	-14.32	-25.71	1.16	-0.0082	-0.00846	0.00022
	15	-8.83	-24.32	2.28	-0.0063	-0.01616	0.00028
f_y	-15	-9.00	-24.31	2.25	-0.0063	-0.01596	0.00028
	-10	-14.44	-25.7	1.14	-0.00821	-0.00831	0.00022
	-5	-19.77	-27.43	0.13	-0.01059	-0.00083	0.00013
	5	-29.77	-31.99	-1.55	-0.01688	0.01313	-0.00018
	10	-34.25	-34.78	-2.19	-0.02071	0.01938	-0.00041
	15	-38.32	-37.8	-2.71	-0.02487	0.02503	-0.00069

Tab. 6.8: Results of estimated parameters of relative position robot-printer for different shift of principal point C and focal length f.

camera ID	t_x **[mm]**	t_y **[mm]**	t_z **[mm]**	$\alpha[rad]$	$\beta[rad]$	$\gamma[rad]$
true value	-20	-20	0	0	0	0
10	-18.58	-27.98	0.18	-0.01130	-0.00248	0.00005
11	-22.52	-31.08	-0.48	-0.01561	0.00303	-0.00024
12	-29.87	-31.73	-1.40	-0.01661	0.01323	0.00017
13	-30.02	-30.24	-1.66	-0.01456	0.01355	0.00038
14	-16.85	-30.26	0.92	-0.01449	-0.00501	-0.00030
15	-25.57	-31.82	-0.42	-0.01661	0.00723	0.00001
16	-29.27	-29.42	-1.26	-0.01340	0.01242	0.00048
17	-22.36	-29.91	-0.26	-0.01398	0.00276	0.00008
18	-25.52	-34.77	-0.25	-0.02064	0.00715	-0.00091
19	-24.10	-29.66	-0.77	-0.01370	0.00533	0.00012
σ	6.21	10.83	0.91	0.01528	0.00832	0.00037
s	4.55	1.83	0.78	0.00251	0.00637	0.00039

Tab. 6.9: Results of robot calibration for different intrinsic parameters.

6.3.4 Different intrinsic camera parameters

Based on results from previous subsection 6.3.3, I measured precision of OpenCV camera resectioning (calibration) in next section 6.4. There I got 10 camera intrinsic parameters sets obtained from randomly chosen images. In this chapter I tested these camera calibrations on one test case (the same as in previous subsection) to see real impact of this inaccuracy of camera resectioning. Results are presented in tab. 6.9, where is shown that sample standard deviation for those 10 camera resectionings is 4.55, 1.83 and 0.78 for translation in x, y and z respectively. For rotation it is 2.51e-3rad, 6.37e-3rad and 0.39e-3rad - around x, y and z axis, respectively. The range of values for translation is almost 14mm for x axis, 7mm for y axis and 2mm for z axis. For rotation it is 0.02rad, 0.18rad and 0.0015rad for α, β and γ.

6.3.5 Different camera models

Since the precise camera resectioning is important part, it is also important to choose suitable camera model. OpenCVSharp provides 3 models as described in sec. 3.1.7 - basic, rational and thin prism model. Unluckily during testing I found out that the rational model alone does not work properly and there is a bug reported ([7]), still unresolved. Therefore, I tested 3 cases of camera models gotten by OpenCV camera resectioning for all 73 images as in sec. 6.4.1 - basic model alone, basic and thin prism model and all three models together. Results in tab. 6.10 shows importance

camera ID	t_x **[mm]**	t_y **[mm]**	t_z **[mm]**	α **[rad]**	β **[rad]**	γ **[rad]**
9	-24.91	-29.53	-0.77	-0.01349	0.00636	0.00000
21	-15.13	-27.10	0.67	-0.01022	-0.00732	0.00029
22	-15.50	-27.72	0.68	-0.01108	-0.00681	0.00025

Tab. 6.10: Results of robot calibration for different camera models. Camera models by camera ID - 9 is basic model, 21 is basic with thin prism model, 22 is basic with rational and thin prism model.

test	**Back-projection error [pixel]**	
	with distortion	**without**
1a	0.7230	0.6039
1b	0.7760	0.5975
2a	0.6354	0.5446
2b	0.8087	0.5430
3a	0.9783	0.6944
3b	0.6030	0.3883
4a	0.8875	0.5915
4b	0.7155	0.5124
Average	0.7659	0.5595

Tab. 6.11: Results of back-projection error for 10 test chessboard cases with and without higher order distortion.

of choosing right model for every camera, because the different model is able to alter robot calibration result by almost 10mm (for t_x) or 0.014rad (for β).

6.3.6 Chessboard Reconstruction

In the same way as marker, I needed to test detection and reconstruction of chessboard using OpenCV method *FindChessboardCorners* and its subpixel method. For this purpose, I created 4 sets of images, each set containing two chessboards. Therefore, I have 8 test cases of chessboard image. To measure accuracy of detection I will use back-projection error again. Results can be seen in tab. 6.11.

6.3.7 Chessboards tests

Since average back-projection error of marker reconstruction is approximately 2 pixels (tab. 6.5) but only 0.5 pixel in case of chessboard reconstruction (tab. 6.11), I wanted to test impact of back-projection error on robot calibration. Therefore, I

test	t_x **[mm]**	t_y **[mm]**	t_z **[mm]**	α **[rad]**	β **[rad]**	γ **[rad]**
true value	-80	30	0	0	0	0
1	-79.207	30.527	7.365	0.00544	-0.00233	0.00126
true value	-100	40	0	0	0	0
2	-104.892	28.421	9.420	-0.01395	0.00684	0.00129
true value	-120	45	0	0	0	0
3	-121.157	41.748	11.559	-0.00434	0.00238	0.00081
true value	-160	60	0	0	0	0
2	-150.427	44.981	17.563	-0.01937	-0.01310	0.00322
σ	5.421	9.624	12.094	0.01243	0.00757	0.00189

Tab. 6.12: Results of chessboards test.

created 4 tests, each consisting of 2 images of chessboard in known relative position. An example of such a test is shown in fig. 6.8.

Fig. 6.8: Example of chessboards test case. From left - original image, first chessboard covered, second chessboard covered.

I use each of two images with one covered chessboard as an argument to RobotToCamera() method (in RobotCalibrator class) and from returned parameters I get final chessboard to chessboard transformation (by calling transform_rbt_to_printer() method in matrixTransform F# module).

As tab. 6.12 shows, standard deviation σ does not differ significantely from results of marker-chessboard tests (tab. 6.7). It is even bigger for some parameters, especially for z-axis, where σ for marker-chessboard test is for translation 2.43mm (in comparison of chessboards test - 12.09mm) and for rotation 0.00044rad (chessboards test - 0.00189rad). Therefore, I can assume that the smaller back-projection error does not influence the result, or at least insignificantly compared to other causes (mainly camera calibration).

images	**accepted images**	C_x **[mm]**	C_y **[mm]**	f_x **[pxl]**	f_y **[pxl]**
30	30	939.6	557.0	4528.5	4558.2
30	30	925.9	577.5	4519.5	4550.5
30	30	945.9	531.9	4522.4	4552.4
30	30	956.9	548.6	4519.5	4550.7
30	30	956.9	564.5	4527.8	4557.3
30	30	972.0	588.1	4511.6	4543.3
30	30	956.0	552.5	4533.7	4564.6
30	30	950.9	575.2	4536.0	4566.3
30	30	968.0	545.8	4525.4	4555.9
30	30	956.0	542.6	4526.8	4557.8
Sample Standard Deviation		13.4	17.7	7.2	6.8

Tab. 6.13: Results of intrinsic parameters calibration tests performed by OpenCV.

6.4 Camera Resectioning

After I tested effect of position of the principal point C and value of focal length f, I needed to test stability (or maybe precision) of camera calibration provided by OpenCV. Because once I do not have precision camera calibration, I am not able to provide precision transform information for robot arm.

6.4.1 OpenCV

As I briefly mentioned in sec. 5.3, the camera class contains method returning intrinsic parameters including distortion coefficients. In this measurement I took over 70 shots of calibration chessboard and then randomly selected 10 sets of 30 photos. Then I got results of calibration for all these sets and compared them.

Camera setting:

- 73 shots
- fully open aperture
- exposure time - 6000μs
- camera ID: 16

Results are in table 6.13.

6.4.2 Matlab

The same test was made with Matlab Camera Calibration Toolbox to eliminate errors in implementation of OpenCV functions and also to compare performance of both algorithms. Results can be seen in tab. 6.14.

images	**accepted images**	C_x **[mm]**	C_y **[mm]**	f_x **[pxl]**	f_y **[pxl]**
30	20	930.7	591.8	4521.0	4549.6
30	24	978.0	592.1	4521.7	4550.3
30	23	956.2	598.1	4527.6	4556.0
30	23	957.5	552.2	4526.8	4557.6
30	21	969.8	586.2	4528.8	4562.0
30	23	946.0	570.6	4500.7	4528.5
30	22	969.5	571.7	4520.0	4547.9
30	24	959.7	569.8	4535.7	4565.4
30	20	933.7	565.5	4529.7	4558.3
30	17	961.1	560.7	4532.0	4561.3
Sample Standard Deviation		15.4	15.3	9.7	10.6

Tab. 6.14: Results of intrinsic parameters calibration tests performed by Matlab.

6.5 Discussion

As shown in sections 6.1 and 6.2, marker detection and camera view calibration are solved with two problems remaining. One is camera position, which is operator responsibility to set it correctly the way that robot arm is not covering part of marker frame in its resting position and from my point of view there is actually nothing to do. If the robot arm covers whole corner shape, then it throws exception that the marker cannot be detected. But in case when the robot arm covers only small part of corner shape, then it is almost impossible to detect this inaccuracy and it leads to inaccurate camera view calibration. The second one is light condition, which is already in progress - custom light is prepared and is waiting only for 5V switch controller.

As I stated at the beginning of section 6.3, camera alone is not able to calibrate robot arm as precisely as needed. Multiple tests were made to find the way how to push this accuracy further. In sections 6.3.1 and 6.3.6 I discovered that additional light does not reduce back-projection error significantly, unlike marker second level refinement which reduces this error in some cases by almost 75%. Similarly I proved that removal of higher order distortion reduces back-projection error from 0.77pixel to 0.56pixel (average for chessboard reconstruction).

Robot arm calibration itself was tested in sec. 6.3. Standard deviation of all tests is not bigger then 7mm for translation (along all axes) and 0.01rad for rotation (around all axes). This could be considered as a good result, but because the smallest buttons are about 5mm wide and robot must touch the screen in precise z-axis coordinate, this is not enough for this case. Therefore, besides back-projection errors

of marker and chessboard reconstruction, I focused on precise camera resectioning. In sections 6.3.3, 6.3.4 and 6.3.5 I proved the importance of getting precise intrinsic parameters for camera and of choosing right camera model. Different camera models shifted result values of translation up to 10mm. As most important intrinsic parameter turned out to be focal length in pixels (f_x and f_y) - by changing their value by ± 15 pixels, translation changes in range up to 30mm and rotation in range up to 0.025rad.

Direct influence of back-projection error of detected points and their model to calibration result was considered in section 6.3.7. Instead of using marker and chessboard in known relative position, I used two chessboards - as chessboard reconstruction provides lower back-projection error. Results in this section showed that the importance of back-projection error reduction is insignificant at least in comparison with precise camera resectioning.

Finally, I tested stability of camera resectioning (camera calibration). I inspected two cases - OpenCV algorithms (sec. 6.4.1) and Matlab camera calibration toolbox (sec. 6.4.1). Both proved to not be stable enough to provide precise robot arm calibration results. Sample standard deviation for crucial parameters f_x and f_y is 7 pixels for OpenCV and 10 pixels for Matlab. With results from sec. 6.3.3 I found out that just by camera resectioning inaccuracy, the result of robot arm calibration can change in range up to 2-3cm.

6.5.1 Optical Hardware

During testing, I tried different camera-lens combinations. Because there is no measurement of these experiments, I will briefly discuss the topic. First, because Basler cameras were sometimes unstable in terms of ethernet connection with the image processing servers, we tried to switch them for Baumer vendor. The state stayed unchanged, so the problem is not on the camera side. During this switch I calibrated Baumer cameras trying to find out if the manufacturing is better. But because of later findings that the calibration itself is relatively unstable, I cannot make any valid conclusion.

Second, I tried lenses with different focal lengths to get camera closer to the device to minimize effect of estimation error of rotation parameters. The problem occurred with devices with smaller screen. The camera got too close to robot arm that it was no longer safe, if robot arm would make some unexpected move. I could solve this by placing the camera not above the robot arm, but in the front of it, meaning also putting it in the front of screen and not above it. This caused troubles with precise detection of robot's chessboard and the whole positioning of camera's tripod was impractical. I wanted to preserve the idea of having only one version of

system, so I kept the original lenses.

Last, I was thinking about changing camera resolution too, but since I found out that the main problem is not the marker or chessboard detection, but precise camera calibration, this would not change the estimation error dramatically so the price for a higher resolution camera would not pay off.

Considering that the team members were not aware about future development of automatic calibration of the whole system in the time they chose the optical equipment, I must state that present system is close to ideal setting due to its positioning, resolution and price. The only significant problem was the overheating due to custom light. I solved this by cutouts on the sides of camera case (seen in fig. 4.10).

7 Conclusion

At the beginning, I analyzed both the assignment and possible solutions. Among many options such as active and passive triangulation, Hough transformation etc., I chose to implement and test one camera reconstruction by iterative methods for robot arm calibration and marker detection for camera view calibration. Unlike other options, this allows to detect printer screen in various light conditions, with different type of screen (resistive/capacitive), with different printer state (idle/on/off), it does not need firm connection between camera and robot arm, it is robust thanks to marker detection and most important - it can be done by the system as it is. Since the accuracy demands were a bit vague - defined the way that robot should be able to tap on the smallest buttons on the screen and also because unknown exact precision of robot movement, I was not able to set any accuracy boundaries for individual algorithm steps (as precision of detection or reconstruction). Even the character of iterative methods, which were used, makes the accuracy demands estimation hard. Therefore, the most straightforward approach was to implement this solution and test its accuracy as one system.

In chapter 3, I explain theory background for methods and algorithm used in solution, from basic topics like RGB-grayscale conversion, 3D->2D projection or contours, to complex algorithm such as Gauss-Newton or Levenberg-Marquardt iterative methods. I also prepared necessary calculations - jacobians for camera position estimation and for sigmoid function fitting, and sigmoid derivative.

In next chapter, I presented solution how to achieve automatic calibration. It is divided into 3 sections - marker detection, camera view calibration and robot arm calibration. In first section, I introduced final marker design (fig.4.2) and its earlier versions. Then I focused on detection itself. I described the whole process (shown in fig. 4.3) in detail covering steps like prefiltering contours, choosing which contour is part of marker and first and second level of refinement. The whole algorithm was made as iterative process with binarization threshold as the parameter which changed through iterations. Here I use prepared sigmoid function - its derivative and jacobian matrix - to maximize detection precision. Then I used this detection to calibrate camera view. It only checks if the detected corner shapes are in the right position with each other and then remove offset of marker frame. Finally, I described robot arm calibration in two sections - arm detection and 3D reconstruction. In the first one, I introduced chessboard holder printed by 3D printer which holds chessboard at the robot arm coordinate system origin. In the reconstruction section, using models of marker and chessboards and their detections, I described the process of getting estimation of parameter vector by Gauss-Newton iterative method. It is done by reconstruction first the printer screen and then the robot arm. From both

estimations I calculate transformation matrix by eq.4.2. Additionally, it turned out that camera calibration is crucial part of the whole process, therefore I described this topic in section 4.4. With that I also designed custom light to enable calibrations during the night (sec. 4.5), also 3D printed.

Chapter 5 gives a brief overview of implementation of chosen solution like programming languages, database model, description of important classes and flow charts.

The most important part of this work is summarized in chapter 6. It consists of different tests, mainly of robot arm calibration and its parts, since this calibration proved to be least accurate. I will only briefly mention some results, for detailed discussion over all results I am redirecting the reader to section 6.5. Marker detection is used for more than half of the year now, mostly without problems. It detects the true presence of marker in more than 94% of all detections (tab.6.2). The biggest problem is light conditions, but the solution is in progress - now it just waits for controller of custom light so it can turn on only if there is marker detection request. Camera view calibration is closely linked with marker detection. Because it mainly depends on precise detection, the results are accurate enough if the marker is detected. If there is unprecise detection due to robot arm or other cause, the camera view is calibrated and superior algorithm decides if it is sufficient. If not, then new calibration is made (and new detection). If there is misdetection of one of corner shapes, then the algorithm is able to detect this and throws an exception (tab.6.3). Robot calibration is tested in terms of marker and chessboard reconstruction, light conditions and intrinsic parameters accuracy. Precision of detection marker and chessboard, and with that also their reconstruction, showed to be less important, unlike camera calibration. To choose precise camera model and get stable results for intrinsic parameters and distortion coefficients turned out to be crucial for robot arm calibration accuracy (sections 6.3.3, 6.3.4 and 6.3.5). With that I tested stability of camera calibration algorithms provided by OpenCV and Matlab. This test proved insufficient stability of both algorithms for this solution. Just because of inaccuracy of getting intrinsic parameters, the result of robot arm calibration can change up to 2-3cm in translation.

With these results I can conclude this - camera view calibration is solved with sufficient accuracy, the same for marker detection. Both have proved to be stable and already saved hundreds of man-hours. Problem with robot arm calibration proved that the system was not able to perform automatic calibration using only camera-based estimation. I suggested 3 solutions. First, to develop extremely precise camera calibration. This means to improve chessboard corner detection and ideally change minimization parameters in Marquardt-Levenberg iterative process, which is mostly used for getting camera parameters [15]. Now the algorithm minimizes back-

projection error of chessboard corners. It would be convenient for me to detect pair of chessboards at one image and minimize not only back-projection error of individual chessboards but also their estimated relative position, specifically its error. Second, to use this estimation only as an initial guess and slowly approach the screen with robot arm and wait for the information from the printer when and where did the arm touch the screen. Third, to use it again as un initial guess but instead of waiting for printer info, use camera to navigate robot arm to a known position (detected known object at the screen - button, icon, text...) in x- and y-axis and using smart stylus, which can tell me when and under what pressure the arm touched something, I can get information about z-axis. By this approach I can touch the screen with the robot arm few times, get information about position of motors and then calibrate the arm. I decided to use the third option and the reason is that the extreme precise camera calibration has no sure result that after its implementation I would not need to use one of the other ones anyway. The connection with the measured device (here printer) is also undesirable so the only possible solution is the third one, which is also the direction of the next development.

Mentioned approach is now in process, the smart stylus, as well as robot arm navigation by camera, is in development and I am expecting to release final solution at the end of summer 2019. This step will finally allow the whole system to be truly operator-free, which opens possibilities like testing more devices with one robot arm at the same time, mobile robotics etc. But mainly, it is major prerequisite for transition from internal YSoft tool to commercial product.

Bibliography

[1] Honec P., Valach S. *Prostorová rekonstrukce*. Dostupné z URL: <http://www.elektrorevue.cz/clanky/04053/index.html>

[2] Efford N. *Image Segmentation*. <https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic3.htm>

[3] Suzuki S., Abe K. *Topological Structural Analysis of Digitized Binary Images by Border Following* Computer Vision, Graphics and Image Processing 30, 32-46, 1985

[4] Duda A., Frese U. *Accurate Detection and Localization of Checkerboard Corners for Calibration*. 2018

[5] OpenCV *Camera Calibration and 3D Reconstruction Library* <https://github.com/opencv/opencv/tree/master/modules/calib3d>

[6] OpenCV *Feature Detection* <https:/docs.opencv.org/2.4/modules/imgproc/doc/faeature_detection.html>

[7] OpenCVSharp *Issue #402* <https://github.com/shimat/opencvsharp/issues/402>

[8] OpenCV *Camera Calibration Tutorial* <https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html>

[9] R. HARTLEY, A. ZISSERMAN, *Multiple View Geometry in Computer Vision, Second Edition*. Cambridge University Press, 2003

[10] Claus D., Fitzgibbon A. W., *A Rational Function Lens Distortion Model for General Cameras*. University of Oxford, 2005

[11] Brown D. C., *Decentering Distortion of Lenses*. D. Brown Associates Inc., 1966

[12] Jung H. G., *Camera Calibration*. Hanyang University, <http://web.yonsei.ac.kr/hgjung/Lectures/AUE859/3.%20Camera%20Calibration.pdf>

[13] Luo Z., Chen Y., Wu S.T., *Wied color gamut LCS with a quantum dot backlight*. Optics Express-November 2013 <https://www.researchgate.net/publication/258444640>

[14] Fajmon B., Hlavičková I., Novák M., *Matematika 3*. Ústav Matematiky FEKT VUT v Brně, 2014

[15] Blaber J., *Camera Calibration Thoery*. justinblaber.org, 2018 <http://justinblaber.org/camera-calibration-theory/>